

全国计算机技术与软件专业技术资格（水平）考试指定用书

数据库系统工程师教程

王亚平 主编 刘强 副主编

全国计算机技术与软件专业技术资格（水平）考试办公室组编

清华大学出版社



全国计算机技术与软件专业技术资格（水平）考试指定用书

数据库系统工程师教程

王亚平 主编 刘强 副主编

全国计算机技术与软件专业技术资格（水平）考试办公室组编

清华大学出版社
北京

www.TopSage.com

全国计算机技术与软件专业资格(水平)考试真题及答案

[2008 年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2008 年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007 年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2007 年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006 年下半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2006 年上半年试题分析与解答 软考指定用书 清华出版](#)(含各科)

[2009 年计算机技术与软件水平考试各科目考试大纲汇总](#)

[全国计算机技术与软件专业资格\(水平\)考试真题及答案汇总](#)

[\[软考视频\]计算机技术与软件专业资格考试推荐视频教程下载汇总](#)

教材及同步辅导见下页。

计算机技术与软件专业技术(水平)考试指定教材及同步辅导

软考初级:

[程序员教程\(第二版\)2007 版 软考指定用书 高清PDF版](#)

[程序员考试辅导: 全国计算机技术与软件专业技术资格\(水平\)考试辅导用书](#)

[网络管理员教程\(第 2 版\)2007 版 软考指定用书 高清PDF版](#)

[网络管理员考试同步辅导\(计算机与网络基础知识篇\) 软考指定辅导用书](#)

[网络管理员考试同步辅导\(网络系统管理与维护篇\) 软考指定使用辅导用书](#)

软考中级:

[网络工程师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[网络工程师教程 软考指定用书 高清PDF版](#)

[网络工程师考试同步辅导: 计算机与网络知识篇 软考指定用书](#)

[网络工程师考试同步辅导\(网络系统设计与管理篇\) 软考指定辅导用书](#)

[软件设计师教程\(第 2 版\) 2007 版 软考指定用书 高清PDF版](#)

[软件设计师考试同步辅导\(下午科目\) 高清PDF版](#)

[软件设计师考试同步辅导\(上午科目\) 高清PDF版](#)

[软件设计师考试考点分析与真题详解\(软件设计技术篇\)](#)

[软件设计师考试辅导：考点精讲、例题分析、强化训练 冶金工业出版](#)

[数据库系统工程师教程 软考指定用书 高清PDF版](#)

[软件评测师教程 软考指定教材 高清PDF版](#)

[信息系统管理工程师教程 软考指定用书 高清PDF版](#)

[信息系统监理师教程 软考指定用书 高清PDF版](#)

软考高级：

[系统分析师教程 软考指定教材 高清PDF版](#)

[系统分析师考试辅导\(2007 版\) 软考指定辅导用书 高清PDF版](#)

[系统分析师教程 PDF文字版](#)

[系统分析师经典教材 Word版](#)

[信息系统项目管理师教程 软考指定教材 高清PDF版](#)

[信息系统项目管理师辅导教程\(上下册\)](#)

[计算机专业英语教程 PDF文字版](#)

更多计算机资料请访问：[大家论坛计算机专区](#)

内 容 简 介

本书按照人事部、信息产业部全国计算机技术与软件专业技术资格(水平)考试的要求编写,内容紧扣《数据库系统工程师考试大纲》。全书共分16章,分别对计算机系统知识、数据结构与算法、操作系统知识、程序语言基础知识、网络基础知识、多媒体基础知识、数据库技术基础、关系数据库、SQL语言、系统开发与运行、数据库设计、数据库运行与管理、网络与数据库、数据库发展趋势与新技术、知识产权基础知识、标准化基础知识进行了系统讲解。

本书内容丰富,语言流畅,层次结构合理,既可供有关考生学习,也可作为培训教材使用。

版权所有,翻印必究。举报电话:010-62782989 13901104297 13801310933

本书扉页为防伪纸、封面贴有清华大学出版社激光防伪标签,无上述标识者不得销售。

图书在版编目(CIP)数据

数据库系统工程师教程/王亚平主编. —北京:清华大学出版社,2004.7

(全国计算机技术与软件专业技术资格(水平)考试指定用书)

ISBN 7-302-09096-3

I. 数… II. 王… III. 数据库系统—工程技术人员—资格考核—教材 IV. TP311.13

中国版本图书馆 CIP 数据核字(2004)第 072800 号

出 版 者: 清华大学出版社 地 址: 北京清华大学学研大厦

<http://www.tup.com.cn> 邮 编: 100084

社 总 机: 010-62770175 客户服务: 010-62776969

组稿编辑: 柴文强

文稿编辑: 徐跃进

印 装 者: 北京嘉实印刷有限公司

发 行 者: 新华书店总店北京发行所

开 本: 185×230 印 张: 46 防伪页: 1 字数: 920 千字

版 次: 2004 年 7 月第 1 版 2004 年 8 月第 2 次印刷

书 号: ISBN 7-302-09096-3/TP·6420

印 数: 30001~38000

定 价: 66.00 元

本书如存在文字不清、漏印以及缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话:(010)62770175-3103 或(010)62795704

序

在国务院鼓励软件产业发展政策的带动下,我国软件业一年一大步,实现了跨越式发展,销售收入由 2000 年的 593 亿元增加到 2003 年的 1633 亿元,年均增长速度 39.2%;2000 年出口软件仅 4 亿美元,去年则达到 20 亿美元,三年中翻了两番多;全国“双软认证工作体系”已经规范运行,截止 2003 年 11 月底,认定软件企业 8582 家,登记软件产品 18 287 个;11 个国家级软件产业基地快速成长,相关政策措施正在落实;我国软件产业的国际竞争力日益提高。

在软件产业快速发展的带动下,人才需求日益迫切,队伍建设与时俱进,而作为规范软件专业技术人员技术资格的计算机软件考试已在我国实施了十余年,累计报考人数超过一百万,为推动我国软件产业的发展作出了重要贡献。

软件考试在全国率先执行了以考代评的政策,取得了良好的效果。为贯彻落实国务院颁布的《振兴软件产业行动纲要》和国家职业资格证书制度,国家人事部和信息产业部对计算机软件考试政策进行了重大改革:考试名称调整为计算机技术与软件专业技术资格(水平)考试;考试对象从狭义的计算机软件扩大到广义的计算机软件,涵盖了计算机技术与软件的各个主要领域(5 个专业类别、3 个级别层次和 20 个职业岗位资格);资格考试和水平考试合并,采用水平考试的形式(与国际接轨,报考不限学历与资历条件),执行资格考试政策(各用人单位可以从考试合格者中择优聘任专业技术职务);这是我国人事制度改革的一次新突破。此外,将资格考试政策延伸到高级资格,使考试制度更为完善。

信息技术发展快,更新快,要求从业人员不断适应和跟进技术的变化,有鉴于此,国家人事部和信息产业部规定对通过考试获得的资格(水平)证书实行每隔三年进行登记的制度,以鼓励和促进专业人员不断接受新知识、新技术、新法规的继续教育。考试设置的专业类别、职业岗位也将随着国民经济与社会发展而动态调整。

目前,我国计算机软件考试的部分级别已与日本信息处理工程师考试的相应级别实现了互认,以后还将继续扩大考试互认的级别和国家。

为规范培训和考试工作,信息产业部电子教育中心组织一批具有较高理论水平和丰富实践经验的专家编写了全国计算机技术与软件专业技术资格(水平)考试的教材和辅导用书,按照考试大纲的要求,全面介绍相关知识与技术,帮助考生学习和备考。

我们相信,经过全社会的共同努力,全国计算机技术与软件专业技术资格(水平)考试将会更加规范、科学,进而对培养信息技术人才,加快专业队伍建设,推动国民经济和社会信息化作出更大的贡献。

敬

信息产业部副部长 姜勤俭

2004年6月

敬爱的姜副部长,您好!我是来自信息产业部计算机与软件专业技术资格考试办公室的姜勤俭副部长,很高兴能与您交流。首先,感谢您一直以来对计算机与软件专业技术资格考试的支持和关注。作为信息产业部副部长,我深知计算机与软件专业技术资格考试在培养信息技术人才、推动国民经济和社会信息化方面的重要作用。因此,我始终关注着考试的发展,并积极与社会各界沟通,共同探讨如何提高考试的质量和水平。

在您的来信中,您提到了关于考试的一些问题,如考试内容的更新、考试形式的多样化等。这些问题都是我们在工作中非常关注的。我们将根据您的建议,认真研究和探讨,力求在下一轮考试中做出改进。同时,我们也希望您能继续关注我们的工作,并提出更多的宝贵意见,帮助我们不断提高考试的质量和水平。

另外,您还提到了关于考试的一些问题,如考试内容的更新、考试形式的多样化等。这些问题都是我们在工作中非常关注的。我们将根据您的建议,认真研究和探讨,力求在下一轮考试中做出改进。同时,我们也希望您能继续关注我们的工作,并提出更多的宝贵意见,帮助我们不断提高考试的质量和水平。

最后,我想再次感谢您一直以来对计算机与软件专业技术资格考试的支持和关注。我们将继续努力,不断提高考试的质量和水平,为培养信息技术人才、推动国民经济和社会信息化做出更大的贡献。

此致
敬礼!

姜勤俭副部长

信息产业部计算机与软件专业技术资格考试办公室

前 言

全国计算机软件考试实施至今已经历了十多年,在社会上产生了很大的影响,对我国软件产业的形成和发展做出了重要的贡献。特别是当今信息化、网络化的时代,计算机信息系统是以数据库为核心的,为了适应我国信息化发展的需求,国家人事部和信息产业部决定将考试的级别拓展到计算机技术与软件各个方面,增设了数据库系统工程师级别的考试,以满足社会上对各种信息技术人才的需要。

编者受全国计算机技术与软件专业技术资格(水平)考试办公室委托,按数据库系统工程师级别的考试大纲要求编写了《数据库系统工程师教程》一书。在考试大纲中,要求考生掌握计算机系统知识、数据库基础知识、数据库及数据库应用系统设计、数据库应用系统实施、数据库系统的运行和管理、网络与数据库、数据库发展及知识产权与标准等方面的知识,涉及的内容多,知识面广,因此编写的难度很高。考虑到参加考试的人员已有一定的基础,所以本书中只对考试大纲中所涉及的知识领域的要点加以阐述,限于篇幅不能详细地展开,请读者谅解。同时,考虑到参加考试的人员一般都已熟悉一种以上的数据库产品和已具有一定的程序设计和编程能力,因此本书主要阐述数据库系统工程师需要掌握的相关内容,对读者在计算机系统知识、数据库技术和应用系统的分析与设计方面起到一定的总结、拓宽和提高的作用,使读者增强数据库及数据库应用系统分析和设计的能力。

全书共分16章,由王亚平担任主编,刘强担任副主编。第1章由李伯成、王亚平编写;第2章由王卫东、张淑平编写;第3章由王亚平编写;第4章由张淑平编写;第5章由张凤琴编写;第6章由刘强编写;第7章由王亚平编写;第8章和第9章由王亚平编写;第10章由褚华编写;第11章和第12章由苏向阳编写;第13章由胡圣明编写;第14章由王小兵编写;第15章和第16章由刘强编写。全书由王亚平统稿。

在本书的编写过程中,参考了许多相关的书籍和资料,编者在此对这些参考文献的作者表示感谢。同时,感谢清华大学出版社在本书出版过程中所给予的支持和帮助。

因水平有限,书中难免存在错漏和不妥之处,望读者指正,以利改进和提高。

编 者
2004年4月

目 录

第 1 章 计算机系统知识	1	2.1.2 数组、矩阵和广义表	71
1.1 计算机系统的组成	1	2.1.3 树	76
1.1.1 计算机发展概述	1	2.1.4 图	87
1.1.2 计算机硬件系统结构	3	2.1.5 查找	99
1.1.3 计算机软件	4	2.1.6 排序	111
1.2 计算机的基本工作原理	5	2.2 常见算法设计方法	125
1.2.1 计算机中数据的表示	5	2.2.1 分治法	125
1.2.2 中央处理机 CPU	10	2.2.2 动态规划	128
1.3 计算机体系结构	12	2.2.3 贪心方法	130
1.3.1 计算机体系结构 的发展	12	2.2.4 回溯法	132
1.3.2 存储系统	15	2.2.5 分支限界法	134
1.3.3 CISC/RISC	24	2.2.6 随机算法	136
1.3.4 输入输出技术	25	2.2.7 近似算法	138
1.3.5 流水线操作	30	第 3 章 操作系统知识	140
1.3.6 总线结构	32	3.1 操作系统基础知识	140
1.3.7 多处理机与并行处理	34	3.1.1 操作系统的定义 与作用	140
1.4 安全性、可靠性与系统性能 评测基础知识	38	3.1.2 操作系统的特征 与功能	141
1.4.1 计算机安全概述	38	3.1.3 操作系统的类型	142
1.4.2 加密技术	40	3.1.4 研究操作系统的观点	145
1.4.3 认证技术	44	3.2 处理机管理	146
1.4.4 计算机病毒的防治	48	3.2.1 基本概念	146
1.4.5 计算机可靠性	54	3.2.2 进程的控制	150
1.4.6 计算机系统的性能 评价	57	3.2.3 进程间的通信	152
1.4.7 计算机故障诊断与 容错	61	3.2.4 管程	155
第 2 章 数据结构与算法	64	3.2.5 进程调度	158
2.1 常用数据结构	64	3.2.6 死锁	159
2.1.1 线性表	64	3.2.7 线程	162
		3.3 存储管理	163
		3.3.1 基本概念	163

3.3.2 分区存储管理	165	基本概念	218
3.3.3 分页存储管理	168	4.1.2 程序设计语言的	
3.3.4 分段存储管理	170	种类与特点	219
3.3.5 段页式存储管理	172	4.1.3 程序设计语言的	
3.3.6 虚拟存储管理	173	基本成分	223
3.4 设备管理	177	4.2 语言处理程序基础	229
3.4.1 设备管理概述	177	4.2.1 汇编程序基本原理	230
3.4.2 I/O 软件	179	4.2.2 编译程序基本原理	233
3.4.3 通道、DMA 与缓冲		4.2.3 解释程序基本原理	260
技术	183	第 5 章 网络基础知识	263
3.4.4 spooling 技术	184	5.1 网络概述	263
3.4.5 磁盘调度	185	5.1.1 计算机网络的概念	263
3.5 文件管理	187	5.1.2 计算机网络的分类	266
3.5.1 文件与文件系统	187	5.1.3 网络的拓扑结构	268
3.5.2 文件的结构和组织	188	5.2 ISO/OSI 网络体系结构	269
3.5.3 文件目录	191	5.3 网络的协议与标准	272
3.5.4 存取方法和存储空间的		5.3.1 网络的标准	273
管理	193	5.3.2 局域网协议	274
3.5.5 文件的使用	195	5.3.3 广域网协议	278
3.5.6 文件的共享和保护	195	5.3.4 Internet 协议	283
3.5.7 系统的安全与可靠性	197	5.4 构建网络	288
3.6 作业与作业管理	199	5.4.1 网络的设备	289
3.6.1 作业管理	199	5.4.2 网络的传输介质	292
3.6.2 作业调度	200	5.4.3 网络的构建	294
3.6.3 用户界面	201	5.5 Internet 及应用	298
3.7 网络操作系统和嵌入式操作系统		5.5.1 Internet 概述	298
基础知识	202	5.5.2 Internet 地址	299
3.7.1 网络操作系统	202	5.5.3 Internet 服务	302
3.7.2 嵌入式操作系统	203	5.6 网络安全	308
3.8 操作系统实例	204	5.6.1 网络安全概述	308
3.8.1 UNIX 操作系统	204	5.6.2 网络的信息安全	310
3.8.2 Windows 2000/XP 操作		5.6.3 防火墙技术	315
系统	212	第 6 章 多媒体基础知识	322
第 4 章 程序设计语言基础	218	6.1 多媒体的基本概念	322
4.1 基础知识	218	6.1.1 媒体的分类	322
4.1.1 程序设计语言的		6.1.2 多媒体的特征	323

6.2 音频	324	7.1.1 数据库与数据库管理系统	365
6.2.1 数字声音基础	324	7.1.2 数据库技术的发展	366
6.2.2 波形声音	326	7.2 数据模型	368
6.2.3 声音合成	328	7.2.1 数据模型的基本概念	368
6.2.4 MIDI	330	7.2.2 数据模型的三要素	369
6.2.5 声音文件格式	331	7.2.3 E-R 模型	370
6.3 图形和图像	332	7.2.4 层次模型	375
6.3.1 色彩与图像基础	332	7.2.5 网状模型	377
6.3.2 计算机中的图形数据表示	334	7.2.6 关系模型	378
6.3.3 图像的获取	335	7.3 DBMS 的功能和特征	380
6.3.4 图像的属性	336	7.3.1 DBMS 的功能	380
6.3.5 图形图像转换	337	7.3.2 DBMS 的特征	381
6.3.6 图像的压缩编码	338	7.4 数据库系统体系结构	382
6.3.7 多媒体数据压缩编码的国际标准	340	7.4.1 数据库的三级模式结构	383
6.3.8 图形、图像文件格式	341	7.4.2 集中式数据库系统	385
6.4 动画和视频	343	7.4.3 客户/服务器数据库体系结构	385
6.4.1 动画	343	7.4.4 并行数据库系统	387
6.4.2 模拟视频	345	7.4.5 分布式数据库系统	388
6.4.3 数字视频	347	7.4.6 Web 数据库	388
6.4.4 数字视频标准	348	7.5 数据库的控制功能	389
6.4.5 视频压缩编码	348	7.5.1 事务管理	389
6.4.6 视频文件格式	350	7.5.2 故障恢复	390
6.5 多媒体网络	352	7.5.3 并发控制	391
6.5.1 超文本与超媒体	352	7.5.4 安全性和授权	393
6.5.2 流媒体的基本概念	353	7.6 数据仓库和数据挖掘基础知识	397
6.5.3 互联网上获取声音和影视的方法	354	7.6.1 数据仓库	397
6.6 多媒体计算机系统	355	7.6.2 数据挖掘	402
6.6.1 多媒体计算机硬件系统	356	第 8 章 关系数据库	406
6.6.2 多媒体软件系统	358	8.1 概述	406
6.7 虚拟现实的概念	362	8.1.1 关系数据库的基本概念	406
第 7 章 数据库技术基础	365	8.1.2 关系数据库模式	408
7.1 基本概念	365		

8.1.3	完整性约束	409	9.4.10	插入、删除和修改 语句	465
8.2	关系运算	410	9.5	SQL 中的授权	466
8.2.1	关系代数运算	410	9.5.1	主键约束 PRIMARY KEY	466
8.2.2	元组演算	422	9.5.2	外键约束 FOREIGN KEY	468
8.2.3	域演算	426	9.5.3	属性值上的约束	468
8.3	查询优化	428	9.5.4	全局约束 CREATE ASSERTIONS	469
8.3.1	基本概念	428	9.5.5	授权与销权	470
8.3.2	关系代数表达式中的 查询优化	428	9.6	触发器	472
8.4	关系数据库设计的基础理论	432	9.6.1	概述	472
8.4.1	基础知识	432	9.6.2	创建触发器	472
8.4.2	规范化	435	9.6.3	删除触发器	474
8.4.3	模式分解及分解应具有 的特性	439	9.7	嵌入式 SQL	475
第 9 章	SQL 语言	446	9.7.1	SQL 与宿主语言接口	475
9.1	数据库语言	446	9.7.2	动态 SQL	478
9.1.1	数据库语言概述	446	9.8	SQL-99 所支持的对象关系 模型	478
9.1.2	数据库语言的分类	447	9.8.1	嵌套关系	478
9.2	SQL 概述	447	9.8.2	复杂类型	481
9.2.1	SQL 语句的特征	447	9.8.3	继承	485
9.2.2	SQL 的基本组成	449	9.8.4	引用类型	488
9.3	数据库定义	449	9.8.5	与复杂类型有关的 查询	488
9.3.1	创建表	449	9.8.6	函数和过程	491
9.3.2	修改表和删除表	450	第 10 章	系统开发与运行	496
9.3.3	定义和删除索引	451	10.1	软件工程和软件开发项目 管理知识	496
9.3.4	定义、删除、更新视图	452	10.1.1	软件工程概述与软件 生存周期	496
9.4	数据操作	454	10.1.2	软件开发项目管理 基础知识	497
9.4.1	Select 基本结构	454	10.1.3	软件开发方法	501
9.4.2	简单查询	455	10.1.4	软件工具与软件开发 环境	502
9.4.3	连接查询	456			
9.4.4	子查询与聚集函数	456			
9.4.5	分组查询	459			
9.4.6	更名运算	460			
9.4.7	字符串操作	461			
9.4.8	集合操作	462			
9.4.9	视图的查询和删除	463			

10.1.5	软件质量管理与质量 保证	504	11.4.2	关系模式的规范化	555
10.1.6	软件过程能力评估	508	11.4.3	确定完整性约束	555
10.2	系统分析基础知识	510	11.4.4	用户视图的确定	555
10.2.1	系统分析概述	510	11.5	数据库的物理设计	556
10.2.2	系统分析方法	511	11.5.1	根据计算机系统的运行 环境进行数据分布	556
10.2.3	系统分析报告	517	11.5.2	确定数据的存储 结构	557
10.3	系统设计知识	519	11.5.3	确定数据的访问 方式	557
10.3.1	系统设计概述	519	11.6	应用程序设计	557
10.3.2	系统总体结构设计	520	11.7	数据库系统的实现	557
10.3.3	系统详细设计	524	11.8	系统实施与维护	558
10.4	系统实施知识	529	11.9	数据库的保护	559
10.4.1	系统实施概述	529	11.9.1	事务的概念	559
10.4.2	程序设计	530	11.9.2	数据库的备份与 恢复	560
10.4.3	系统测试与调试	532	11.9.3	数据库的安全性	563
10.4.4	系统文档	537	11.9.4	数据库的完整性	563
10.4.5	系统转换	538	11.9.5	数据库的并发控制	564
10.5	系统运行和维护知识	539	11.10	小结	566
10.5.1	系统维护	539	第 12 章	数据库运行与管理	567
10.5.2	系统评价	541	12.1	数据库系统的运行计划	567
10.5.3	系统运行管理	542	12.1.1	运行策略的确定	567
第 11 章	数据库设计	546	12.1.2	确定数据库系统监控 对象和监控方式	568
11.1	数据库设计概述	546	12.1.3	数据库系统管理 计划	569
11.2	系统需求分析	547	12.2	数据库系统的运行和维护	569
11.2.1	需求分析的任务和 目标	547	12.2.1	监控数据的收集与 分析	569
11.2.2	需求分析的方法和 步骤	548	12.2.2	稳定运行中的业务 持续性	569
11.3	概念结构设计	550	12.2.3	数据库维护	570
11.3.1	概念结构设计策略与 方法	551	12.2.4	数据库系统的运行 统计	571
11.3.2	用 E-R 方法建立概念 模型	552			
11.4	逻辑结构设计	553			
11.4.1	E-R 图向关系模式的 转换	554			

12.2.5 数据库系统的审计	571	13.2.4 动态 Web 网页	604
12.3 数据库系统的管理	571	13.2.5 CGI 的应用	605
12.3.1 数据字典的管理	571	13.2.6 ASP 的应用	606
12.3.2 数据完整性维护和 管理	572	13.2.7 Servlet 和 JSP 的应用	608
12.3.3 数据库的存储管理	572	13.3 XML 与数据库	609
12.3.4 备份和恢复	572	13.3.1 什么是 XML	609
12.3.5 并发控制与死锁 管理	573	13.3.2 XML 文件存储面临的 问题	610
12.3.6 数据安全性管理	573	13.3.3 XML 与数据库的数据 转换	611
12.4 性能调整	573	第 14 章 数据库发展趋势与新技术	616
12.4.1 SQL 语句的编码 检验	573	14.1 面向对象数据库	616
12.4.2 表设计的评价	574	14.1.1 面向对象数据库系统的 特征	617
12.4.3 索引改进	574	14.1.2 面向对象数据模型	618
12.4.4 设备增强	574	14.1.3 面向对象数据库 语言	622
12.5 用户支持	575	14.1.4 对象关系数据库 系统	623
12.5.1 用户培训	575	14.2 ERP 和数据库	630
12.5.2 售后服务	575	14.2.1 ERP 概述	630
12.6 小结	576	14.2.2 ERP 与数据库	638
第 13 章 网络与数据库	577	14.2.3 案例分析	640
13.1 分布式数据库	577	14.3 决策支持系统的建立	644
13.1.1 分布式数据库 的概念	578	14.3.1 决策支持系统 的概念	644
13.1.2 分布式数据库的体系 结构	581	14.3.2 数据仓库设计	646
13.1.3 分布式查询处理和 优化	590	14.3.3 数据转移技术	649
13.1.4 分布事务管理	591	14.3.4 OLAP 技术	653
13.1.5 分布式数据库系统的 应用	599	14.3.5 企业决策支持解决 方案	657
13.2 Web 与数据库	599	14.3.6 联机事务处理	662
13.2.1 Web 概述	600	第 15 章 知识产权基础知识	665
13.2.2 Web 服务器脚本程序与 服务器的接口	602	15.1 知识产权的概念与特点	665
13.2.3 应用开发平台	603	15.1.1 知识产权的概念	665

15.1.2	知识产权的特点	666	15.6	计算机软件著作权侵权	
15.1.3	我国保护知识产权的			的鉴别	677
	法规	668	15.6.1	计算机软件著作权	
15.2	计算机软件著作权的主体与			侵权行为	678
	客体	668	15.6.2	不构成计算机软件侵权的	
15.2.1	计算机软件著作权的			合理使用行为	679
	主体	668	15.6.3	计算机著作权软件	
15.2.2	计算机软件著作权的			侵权的识别	680
	客体	669	15.7	软件著作权侵权的法律责任	681
15.3	计算机软件受著作权法保护的		15.8	计算机软件的商业秘密权	682
	条件	670	15.8.1	商业秘密的概念	682
15.4	计算机软件著作权的权利	670	15.8.2	计算机软件商业秘密的	
15.4.1	计算机软件的著作			侵权	683
	人身权	670	15.8.3	计算机软件商业秘密侵权	
15.4.2	计算机软件的著作			的法律责任	684
	财产权	671	15.9	专利权概述	685
15.4.3	软件合法持有人的		15.9.1	专利权的保护对象与	
	权利	672		特征	685
15.4.4	计算机软件著作权的		15.9.2	授予专利权的条件	685
	行使	672	15.9.3	专利的申请	686
15.4.5	计算机软件著作权的		15.9.4	专利权的行使	688
	保护期	673	15.9.5	专利权的限制	688
15.5	计算机软件著作权的归属	673	15.9.6	专利侵权行为	689
15.5.1	软件著作权归属的		15.10	企业知识产权的保护	690
	基本原则	673	15.10.1	知识产权管理	690
15.5.2	职务开发软件著作权的		15.10.2	知识产权的保护和	
	归属	673		利用	690
15.5.3	合作开发软件著作权的		15.10.3	建立经济约束机制规范	
	归属	674		调整各种关系	691
15.5.4	委托开发的软件著作权		第 16 章	标准化基础知识	693
	归属	675	16.1	标准化的基本概念	693
15.5.5	接受任务开发的软件著作		16.1.1	标准、标准化	
	权归属	676		的概念	693
15.5.6	计算机软件著作权主体		16.1.2	标准化的范围	
	变更后软件著作权的			和对象	693
	归属	676	16.1.3	标准化的实质	694

16.1.4	标准化的目的	695	16.6	信息技术标准化	707
16.2	标准化过程模式	695	16.6.1	信息编码标准化	707
16.2.1	标准的制定	695	16.6.2	条码标准化	707
16.2.2	标准的实施	696	16.6.3	汉字编码标准化	708
16.2.3	标准的更新	696	16.6.4	软件工程标准化	708
16.3	标准的分类	697	16.7	标准化组织	709
16.3.1	根据适用范围分类	697	16.7.1	国际标准化组织	709
16.3.2	根据标准的性质 分类	699	16.7.2	区域标准化组织	711
16.3.3	根据标准化的对象和 作用分类	700	16.7.3	行业标准化组织	712
16.3.4	根据法律的约束性 分类	701	16.7.4	国家标准化组织	712
16.4	标准的代号和编号	702	16.8	ISO9000 标准简介	713
16.5	国际标准和国外先进标准	703	16.8.1	ISO9000 标准	713
16.5.1	国际标准	704	16.8.2	ISO9000;2000 系列标准 文件结构	714
16.5.2	国外先进标准	704	16.8.3	ISO9000;2000 核心标准 简介	714
16.5.3	采用国际标准和国外 先进标准	704	16.8.4	ISO9000;2000 系列标 准确认的 8 项原则	715
16.5.4	采用程度的概念	705	16.9	能力成熟度模型 CMM 简介	717
16.5.5	采用国际标准和国外 先进标准的原则	706	16.10	ISO/IEC 15504 过程评估标准 简介	719

第 1 章 计算机系统知识

自 1946 年第一台计算机 ENIAC(Electronic Numerical Integrator and Calculator)问世以来,计算机的发展异常迅速,从单一的数值处理发展到非数字处理和多媒体信息处理;从科学计算领域发展到商业、办公、学习和日常生活领域;从早期的以运算器为中心的冯·诺依曼结构发展到流水线、并行处理和多处理机结构;从传统的指令驱动型计算发展到数据驱动和需求驱动型计算,可谓日新月异。今天处处可以看到计算机对现代社会带来的深刻影响,计算机的发明和应用,在人类文明史上具有划时代的历史意义。

1.1 计算机系统的组成

1.1.1 计算机发展概述

计算机的发明和应用是 20 世纪人类最重要的成就,标志着信息时代的开始。在此后的近 50 年里,计算机得到飞速的发展,使得计算机及其应用已经渗透到社会的各个领域,有力地推动了社会信息化发展。目前,一个国家计算机的应用水平直接标志着一个国家的科学现代化水平。计算机的发展经历了 5 个重要的阶段。

1. 大型机阶段

1946 年美国研制的第一台计算机 ENIAC 被公认为大型机的鼻祖。它采用电子管制作计算机的基本逻辑部件,体积大,耗电量大,寿命短,可靠性差,成本高,而且由于采用电子射线管作为计算机的存储部件,所以容量很小。

大型机(mainframe)的发展经历了以下几代:

- 第一代,采用电子管制作计算机;
- 第二代,采用晶体管制作计算机;
- 第三代,采用中、小规模集成电路制作计算机;
- 第四代,采用大规模、超大规模集成电路制作计算机,其代表机型有 IBM 360/370/709/4300/9000 等。

2. 小型机阶段

小型机(minicomputer)或称小型电脑,通常用来满足部门的需要,被中小型企业事业单位使用,例如,DEC 公司的 VAX 系列机。

3. 微型机阶段

微型机(microcomputer)又称微电脑或个人电脑(personal computer, PC)。顾名思义,该机是面向个人或家庭的,它的价格与高档家用电器相当,应用相当普及,例如,Apple II、IBM-PC 系列机。

4. 客户机/服务器阶段

1964 年美国航空公司建立了第一个联机订票系统,将全美的 2000 个订票终端用电话线连在一起。订票中心的大型机,即服务器,用来处理订票事务;而分散在各地的订票终端则称为客户机。从逻辑上来看,这是早期客户机/服务器模式。

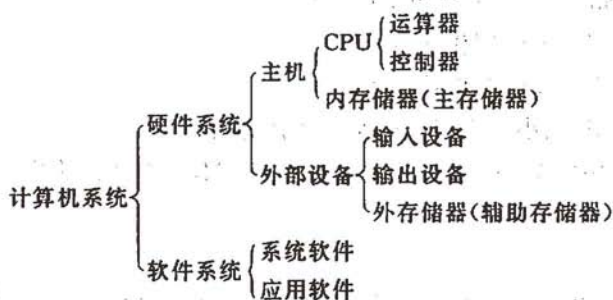
早期的服务器主要是为客户机提供资源共享的磁盘服务器和文件服务器,而现在的服务器主要是数据库服务器和应用服务器等。

客户机/服务器(Client/Server)模式是对大型机的一次挑战。由于客户机/服务器模式的结构灵活,适应面广,成本较低,因此得到了广泛的应用。如果服务器的处理能力强,客户机的处理能力弱,则称为瘦客户机/胖服务器;否则称之为胖客户机/瘦服务器。

5. 互联网阶段

自 1969 年美国国防部 ARPANET 网运行以来,计算机广域网开始发展起来。1983 年 TCP/IP(传输控制与互联网协议)正式成为 ARPANET 网的标准协议,这使得网际互联有了突飞猛进的发展。以它为主干发展起来的因特网(Internet)到 1990 年已连接 3000 多个网络和 20 万台计算机。进入 20 世纪 90 年代,因特网继续以指数级速度迅猛扩展。进入 21 世纪,全球有上亿因特网用户。到 1994 年,我国采用 TCP/IP 协议通过 4 大主干网接入因特网。目前全国的因特网用户数已超过 8000 万。

计算机系统是由硬件系统和软件系统组成的。计算机硬件是计算机系统中看得见、摸得着的物理装置,计算机软件是程序、数据和相关文档的集合。计算机系统的组成如下所示。



1.1.2 计算机硬件系统结构

1. 计算机的硬件组成

计算机硬件由运算器、控制器、存储器、输入设备和输出设备 5 大部件组成,如图 1-1 所示。随着技术的发展,运算器、控制器等部件已被集成在一起,统称为中央处理单元(central processing unit,CPU)。它是硬件系统的核心,用于数据的加工处理,能完成各种算术、逻辑运算及控制功能。存储器是计算机系统记忆设备,分为内部存储器和外部存储器。前者速度高,容量小,一般用于临时存放程序、数据及中间结果。而后者容量大,速度慢,可以长期保存程序和数据。输入设备和输出设备合称为外部设备(简称外设)。输入设备用于输入原始数据及各种命令,而输出设备则用于输出计算机运行的结果。

运算器是对数据进行加工处理的部件,它主要完成算术和逻辑运算。控制器的主要功能是从主存中取出指令,并指出下一条指令在主存中的位置。取出的指令经指令寄存器送往指令译码器,经过对指令的分析发出相应的控制和定时信息,控制计算机的各个部件有条不紊地工作,以完成指令所规定的操作。寄存器是计算机系统记忆设备,用来存放程序、原始数据、中间结果及最终结果。

输入设备的作用是把程序和原始数据转换成计算机中表示的二进制数,输入到计算机的主存中。输出设备的作用是把运算处理结果按照人们所要求的形式输出到外部存储介质上。

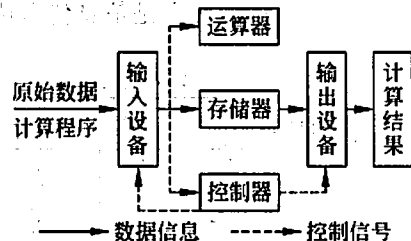


图 1-1 计算机组成框图

2. 计算机硬件的典型结构

(1) 单总线结构: 图 1-2 是单总线的计算机系统结构,即用一组系统总线将计算机系统的各部件连接起来,各部件之间可以通过总线交换信息。这种结构的优点是易于扩充新的 I/O 设备,并且各种 I/O 设备的寄存器和主存储器的存储单元可以统一编址,使 CPU 访问 I/O 设备更方便灵活;其缺点是同一时刻只能允许挂在总线上的一对设备之间互相传送信息,也即分时使用总线,这就限制了信息传送的吞吐量。这种结构一般用在微型计算机和小型计算机中。

(2) 双总线结构: 为了消除信息传送的瓶颈,常设置多组总线,最常见的是在主存和 CPU 之间设置一组专用的高速存储总线,如图 1-3 所示。图 1-3(a)是以 CPU 为中心的双总线结构,图 1-3(b)是以存储器为中心的双总线结构。在以 CPU 为中心的双总线结构中,将连接 CPU 和外围设备的系统总线称为输入输出(I/O)总线。这种结构的优点是

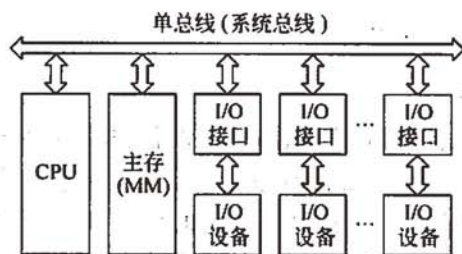
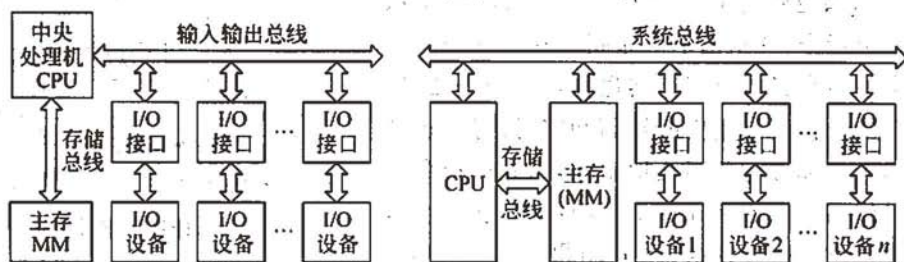


图 1-2 单总线计算机系统结构

控制线路简单,对 I/O 总线的传送速率要求较低;其缺点是 CPU 的工作效率较低,因为 I/O 设备与主存之间的信息交换要经过 CPU 进行。在以存储器为中心的双总线结构中,主存储器可通过存储总线与 CPU 交换信息,同时还可以通过系统总线与 I/O 设备交换信息。这种结构的优点是信息传送速率高;其缺点是需要增加硬件的投资。



(b) 以存储器为中心的双总线组成结构

图 1-3 双总线结构

(3) 采用通道的大型系统结构:为了扩大系统的功能和提高系统的效率,在大、中型计算机系统中采用通道结构,如图 1-4 所示。

在这种结构中,一台主机可以连接多个通道,一个通道可以连接一台或多台 I/O 控制器,一台 I/O 控制器又可以连接一台或多台 I/O 设备,所以它具有较大的扩展余地。另外,由通道来管理和控制 I/O 设备,减轻了 CPU 的负担,提高了整个系统的效率。

1.1.3 计算机软件

在计算机系统中如果仅有硬件系统,那么只具备了计算的功能,并不能真正运算,只有将解决问题的步骤编制成程序,并由输入设备输入到计算机内存中,通过系统软件的支持,才能完成运算。软件是指为管理、运行、维护及应用的计算机所开发的程序和相关文档的集合。可见,计算机系统除了硬件系统,还必须有软件系统。软件系统是计算机系统

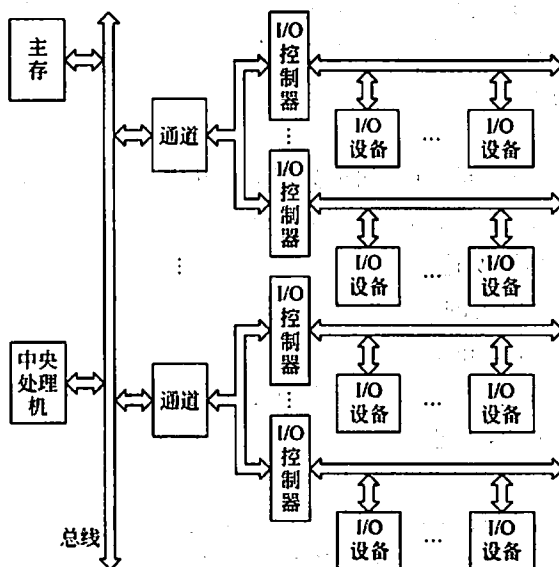
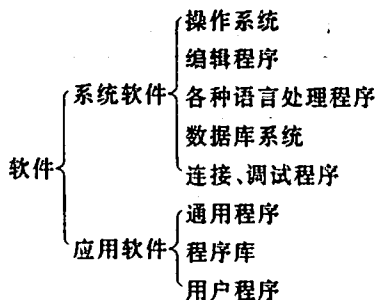


图 1-4 大型计算机系统的通道结构

中的重要组成部分,通常可将软件分为两大类:系统软件和应用软件,如下所示:



1.2 计算机的基本工作原理

1.2.1 计算机中数据的表示

计算机最主要的功能是处理信息,如处理数值、文字、声音、图形和图像等。在计算机内部,各种信息都必须经过数字化编码后才能被传送、存储和处理。因此,掌握信息编码的概念与处理技术是至关重要的。所谓编码,就是采用少量的基本符号,选用一定的组合

原则,以表示大量复杂多样的信息。基本符号的种类和这些符号的组合规则是一切信息编码的两大要素。例如,用 10 个阿拉伯数码表示数字,用 26 个英文字母表示英文词汇等,都是编码的典型例子。

1. 进位记数制

在采用进位计数的数字系统中,如果只用 r 个基本符号(例如, $0, 1, 2, \dots, r-1$)表示数值,则称其为基 r 数制(radix- r number system), r 称为该数制的基(radix)。对于不同的数制,它们的共同特点如下所述。

- 每一种数制都有固定的符号集,如十进制数制,其符号有 10 个: $0, 1, \dots, 9$ 。二进制数制,其符号有 0 和 1 两个。
- 其次,都使用位置表示法,即处于不同位置的数符所代表的值不同,与它所在位置的权值有关。例如,十进制数 1234.55 可表示为:

$$1234.55 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1} + 5 \times 10^{-2}$$

可以看出,各种进位记数制中的权的值恰好是基数的某次幂。因此,对任何一种进位记数制表示的数都可以写成按权展开的多项式之和,任意一个 r 进制数 N 可表示为

$$N = \sum_{i=m-1}^{-k} D_i \times r^i$$

式中的 D_i 为该数制采用的基本数符, r^i 是权, r 是基数,不同的基数,表示不同的进制数。表 1-1 所示的是计算机中常用的几种进位数制。

表 1-1 计算机中常用的几种进制

进位制	二进制	八进制	十进制	十六进制
规则	逢二进一	逢八进一	逢十进一	逢十六进一
基数	$r=2$	$r=8$	$r=10$	$r=16$
数符	0, 1	0, 1, ..., 7	0, 1, ..., 9	0, 1, ..., 9, A, B, ..., F
权	2^i	8^i	10^i	16^i
缩写形式	B	O	D	H

2. 算术逻辑运算

(1) 二进制加法: 二进制加法与十进制加法相类似,所不同的是,二进制加法的规则是“逢二进一”,即:

$$0+0=0 \quad 1+0=1 \quad 0+1=1 \quad 1+1=0(\text{有进位})$$

(2) 二进制减法: 在二进制减法中,当不够减时需要借位,高位的 1 等于下一位的 2,即“借一当二”,其运算法则如下:

$$0-0=0 \quad 1-0=1 \quad 1-1=0 \quad 0-1=1(\text{有借位})$$

(3) 二进制乘法：二进制乘法与十进制乘法是一样的。但因为二进制数只由 0 和 1 构成，因此，二进制乘法更简单。其法则如下：

$$0 \times 0 = 0 \quad 1 \times 0 = 0 \quad 0 \times 1 = 0 \quad 1 \times 1 = 1$$

(4) 二进制除法：二进制除法是乘法的逆运算，其运算方法与十进制除法是一样的。

(5) 二进制与运算又称逻辑乘，其法则为：

$$0 \wedge 0 = 0 \quad 0 \wedge 1 = 0 \quad 1 \wedge 0 = 0 \quad 1 \wedge 1 = 1$$

(6) 二进制或运算又称逻辑加，其法则为：

$$0 \vee 0 = 0 \quad 0 \vee 1 = 1 \quad 1 \vee 0 = 1 \quad 1 \vee 1 = 1$$

(7) 二进制异或的法则为：

$$0 \oplus 0 = 0 \quad 0 \oplus 1 = 1 \quad 1 \oplus 0 = 1 \quad 1 \oplus 1 = 0$$

3. 机器数和码制

各种数据在计算机中表示的形式称为机器数。其特点是数的符号用 0、1 表示，如“0”表示正号，“1”表示负号；小数点则隐含表示而不占位置。机器数对应的实际数值称为该数的真值。

机器数有无符号数和带符号数两种。无符号数表示正数，在机器数中没有符号位。对于无符号数，若约定小数点的位置在机器数的最低位之后，则是纯整数；若约定小数点的位置在机器数的最高位之前，则是纯小数。对于带符号数，机器数的最高位是表示正、负的符号位，其余二进制位表示数值。若约定小数点的位置在机器数的最低数值位之后，则是纯整数；若约定小数点的位置在机器数的最高数值位之前（符号位之后），则是纯小数。

为了便于运算，带符号的机器数可采用原码、反码和补码等不同的编码方法，机器数的这些编码方法称为码制。

4. 汉字编码

汉字处理包括汉字的编码输入、汉字的存储和汉字的输出等环节。也就是说计算机处理汉字，首先必须先将汉字代码化，即对汉字进行编码。无论西方的拼音文字还是汉字这种象形文字，它们的“意”都寓于它们的“形”和“音”上。前面介绍过，直接向计算机输入文字的字形和语音虽然可以实现，但还不够理想。在计算机内部直接处理，存储文字的字形和语音就更困难了，所以用计算机处理字符，尤其是处理汉字字符，一定要把字符代码化。

西文是拼音文字，基本符号比较少，编码比较容易，而且在一个计算机系统中，输入、内部处理、存储和输出都可以使用同一代码。汉字种类繁多，编码比拼音文字困难，而且在一个汉字处理系统中，输入、内部处理、存储和输出对汉字代码的要求不尽相同，所以用的代码也不尽相同。汉字信息处理系统在处理汉字和词语时，关键的问题是要进行一系

列的汉字代码转换。

1) 输入码

中文的字数繁多,字形复杂,字音多变,常用汉字就有 7000 个左右。在计算机系统中使用汉字,首先遇到的问题就是如何把汉字输入到计算机内。为了能直接使用西文标准键盘进行输入,必须为汉字设计相应的编码方法。汉字编码方法主要分为三类:数字编码,拼音码和字形码。

(1) 数字编码。数字编码就是用数字串代表一个汉字的输入,常用的是国标区位码。国际区位码将国家标准局公布的 6763 个两级汉字分成 94 个区,每个区 94 位。它实际上把汉字表示成二维数组,区位和位码各为两位十进制数字,因此,输入一个汉字需要按键四次。例如,“中”字位于第 54 区 48 位,区位码为 5448。

汉字在区位码表中的排列是有规律的。在 94 个分区中,1~15 区用来表示字母、数字和符号,16~87 区为一级和二级汉字。一级汉字以汉语拼音为序排列,二级汉字以偏旁部首进行排列。使用区位码方法输入汉字时,必须先在表中查找汉字并找出对应的代码,才能输入。数字编码输入的优点是无重码,而且输入码和内部编码的转换比较方便,但是每个编码都是等长的数字串,代码难以记忆。

(2) 拼音码。拼音码是以汉语读音为基础的输入方法。由于汉字同音字太多,输入重码率很高,因此,按拼音输入后还必须进行同音字选择,影响了输入速度。

(3) 字形编码。字形编码是以汉字的形状确定的编码。汉字总数虽多,但都是由一笔一划组成的,全部汉字的部件和笔划是有限的。因此,把汉字的笔划部件用字母或数字进行编码,按笔划书写的顺序依次输入,就能表示一个汉字。五笔字型、表形码等便是这种编码法。五笔字形编码是最有影响的编码方法。

2) 内部码

汉字内部码(简称汉字内码)是汉字在设备或信息处理系统内部最基本的表达形式,是在设备和信息处理系统内部存储、处理、传输汉字用的代码。在西文计算机中,没有交换码和内码之分。汉字数量多,用一个字节无法区分,采用国家标准局 GB2312-80 中规定的汉字国标码,两个字节存放一个汉字的内码,每个字节的最高位置“1”,作为汉字机内码。由于两个字节各用 7 位,因此可表示 16 384 个可区别的机内码。以汉字“大”为例,国标码为 3473H,两个字节的最高位置“1”,得到的机内码为 B4F3H。

为了统一地表示世界各国的文字,1993 年国际标准化组织公布“通用多八位编码字符集”的国际标准 ISO/IEC 10646,简称 UCS(Universal Code Set)。UCS 包含了中、日、韩等国的文字,这一标准为包括汉字在内的各种正在使用的文字规定了统一的编码方案。该标准是用 4 个 8 位码(4B)来表示每个字符的,并相应地指定组、平面、行和字位。

- 组：用一个 8 位二进制数来编码组，最高位不用，剩下 7 位，能表示 128 个组。
- 平面：用一个 8 位二进制来编码平面，能表示 256 平面，这样每一组包含 256 平面。
- 行：用一个 8 位二进制来编码行，能表示 256 行，这样每一平面包含 256 行。
- 字位：用一个 8 位二进制数来编码字位，能表示 256 字位，这样每一行包含 256 个字位。
- 一个字符就被安排在这个编码空间的一个字位上。4 个 8 位码，即 32 位二进制数足以包容世界上所有的字符，同时也符合现代处理系统的体系结构。

第一个平面(00 组中的 00 平面)称为基本多文种平面。它包含字母文字、音节文字及表意文字等，分成 4 个区。

(1) A 区：代码位置 0000H~4DFFH(19903 个字位)用于字母文字、音节文字以及各种符号。

(2) I 区：代码位置 4E00H~9FFFH(20992 个字位)用于中、日、韩(CJK)统一的表意文字。

(3) O 区：代码位置 A000H~DFFFH(16384 个字位)留作未来标准化用。

(4) R 区：代码位置 E000H~FFFDH(8190 个字位)作为基本多文种平面的限制使用区，它包括专用字符、兼容字符等。

例如：

ASCII 字符“A”，它的 ASCII 码为 41H。它在 UCS 中的编码为 00000041H，即在 00 组、00 面、00 行、第 41H 字位上。

汉字“大”，它在 GB2312 中的编码为 3474H，它在 UCS 中的编码为 00005927H，即在 00 组、00 面、59H 行、第 27H 字位上。

我国相应的国家标准为 GB13000。详细内容请查阅网址：<http://www.unicode.org>。

3) 字形码

汉字字形码是表示汉字字形的字模数据，通常用点阵、矢量函数等方式表示。用点阵表示字形时，汉字字形码指的就是这个汉字字形点阵的代码。字形码也称字模码，是用点阵表示的汉字字形码，它是汉字的输出方式。根据输出汉字的要求不同，点阵的多少也不同。简易型汉字为 16×16 点阵，高精度型汉字为 24×24 点阵、32×32 点阵、48×48 点阵等。

字模点阵的信息量是很大的，所占存储空间也很大。以 16×16 点阵为例，每个汉字就不能用于机内存储。字库中存储了每个汉字的点阵代码，当显示输出时才检索字库，输出字模点阵得到字形。

汉字的矢量表示法是将汉字看做由笔画组成的图形,提取每个笔画的坐标值,这些坐标值就可以决定每一笔画的位置。将每一个汉字的所有坐标值信息组合起来就是该汉字字形的矢量信息。显然,汉字的字形不同其矢量信息也就不同,每个汉字都有自己的矢量信息。由于汉字的笔画不同,所以矢量信息就不同,每个汉字矢量信息所占的内存大小不一样。这与点阵表示法不一样。

同样,将每一个汉字的矢量信息集中在一起就构成了汉字库。当需要汉字输出时,利用汉字字形检索程序并根据汉字内码从字模库中找到相应的字形码。

1.2.2 中央处理机 CPU

1. CPU 的组成

前面已经提到,CPU 主要由运算器、控制器组成。构成 CPU 的框图如图 1-5 所示。

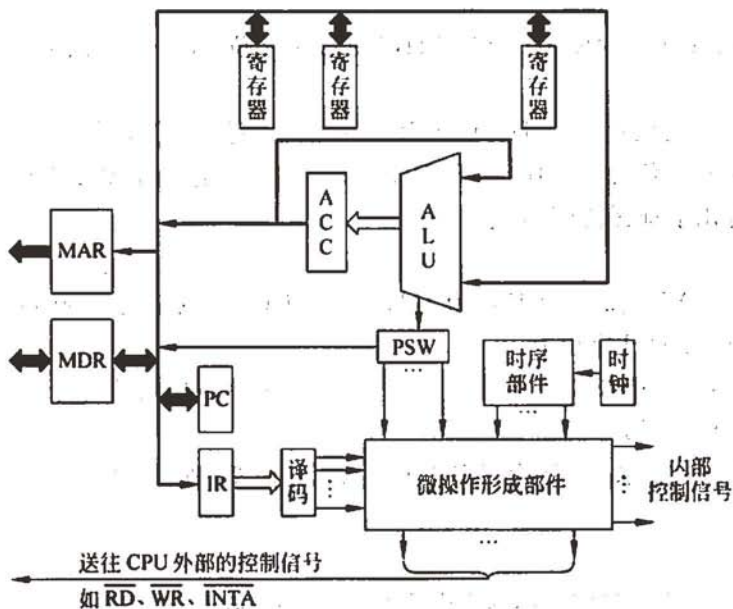


图 1-5 CPU 组成框图

1) 运算器

运算器是对数据进行加工处理的部件。它主要完成算术运算和逻辑运算,完成对数据的加工与处理。不同的计算机,运算器的结构也不同,但最基本的结构都由算术/逻辑运算单元(ALU)、累加器 ACC、寄存器组、多路转换器和数据总线等逻辑部件组成。带多路选择器的运算器结构如图 1-6 所示。

2) 控制器

计算机能执行的基本操作叫做指令，一台计算机的所有指令组成指令系统。指令由操作码和地址码两部分组成，操作码指明操作的类型，地址码则指明操作数及运算结果存放的地址。

控制器的主要功能是从内存中取出指令，并指出下一条指令在内存中的位置。将取出的指令经指令寄存器送往指令译码器，经过对指令的分析发出相应的控制和定时信息，控制和协调计算机的各个部件有条不紊地工作，以完成指令所规定的操作。

控制器是由程序计数器(简称 PC)、指令寄存器、指令译码器、状态条件寄存器、时序产生器、微操作信号发生器组成，如图 1-7 所示。其主要作用如下所述。

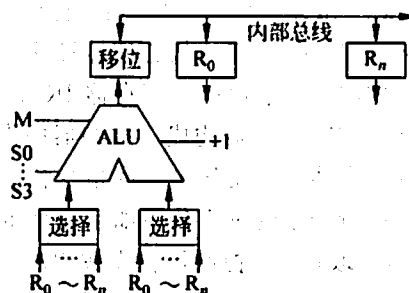


图 1-6 带多路选择器的运算器

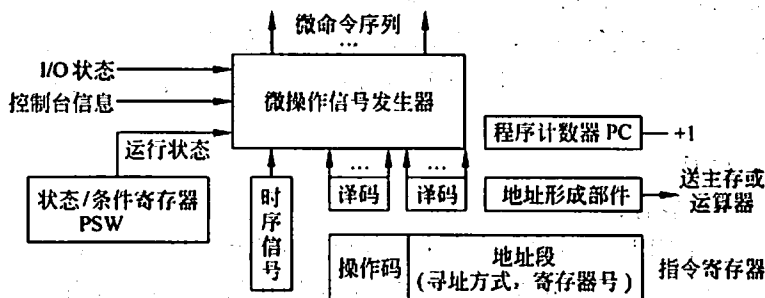


图 1-7 控制器组成框图

(1) 程序计数器：当程序顺序执行时，每取出一条指令，PC 内容自动增加一个值，指向下一条要取的指令。当程序出现转移时，则将转移地址送入 PC，然后由 PC 指向新的程序地址。

(2) 指令寄存器(IR)：用于存放当前要执行的指令。

(3) 指令译码器(ID)：对现行指令进行分析，确定指令类型、指令所要完成的操作以及寻址方式。

(4) 时序产生器：用于产生时序脉冲和节拍电位去控制计算机有序地工作。

(5) 状态/条件寄存器：用于保存指令执行完成后产生的条件码，例如运算是否有溢出，结果为正还是为负，是否有进位等。此外，状态/条件寄存器还保存中断和系统工作状态等信息。

(6) 微操作信号发生器：把指令提供的操作信号、时序产生器提供的时序信号及由控制功能部件反馈的状态信号等综合成特定的操作序列，从而完成取指令的执行控制。

控制器一般由指令寄存器 IR、程序计数器 PC、时序部件、微操作形成部件和程序状态字(PSW)寄存器构成。控制器的作用是控制整个计算机的各个部件有条不紊地工作，它的基本功能就是从内存取指令和执行指令。

执行指令的过程分为如下几个步骤。

① 取指令：控制器首先按程序计数器所指出的指令地址从内存中取出一条指令。

② 指令译码：将指令的操作码部分送入指令译码器进行分析，然后根据指令的功能向有关部件发出控制命令。

③ 按指令操作码执行：根据指令译码器分析指令产生的操作控制命令以及程序状态字(PSW)寄存器的状态，控制微操作形成部件产生一系列 CPU 内部的控制信号和输出到 CPU 外部的控制信号。在这一系列控制信号的控制下，实现指令的具体功能。

④ 形成下一条指令地址：若非转移类指令，则修改指令地址寄存器的内容；若是转移类指令，则根据转移条件修改指令地址寄存器的内容。

通过上述步骤逐一执行一系列指令，就使计算机能够按照这一系列指令组成的程序的要求自动完成各项任务。控制器和运算器合在一起被称为中央处理单元，即 CPU，它是计算机的核心。

2. CPU 的功能

CPU 的基本功能有：

(1) 程序控制：CPU 通过执行指令来控制程序的执行顺序，这是 CPU 的重要职能。

(2) 操作控制：一条指令功能的实现需要若干操作信号来完成，CPU 产生每条指令的操作信号并将操作信号送往不同的部件，控制相应的部件按指令的功能要求进行操作。

(3) 时间控制：CPU 对各种操作进行时间上的控制，这就是时间控制。CPU 对每条指令整个的执行时间要进行严格控制。同时，指令执行过程中的操作信号的出现时间、持续时间及出现的时间顺序都需进行严格控制。

(4) 数据处理：CPU 对数据进行算术运算及逻辑运算等方式进行加工处理，数据加工处理的结果为人们所利用。所以，对数据的加工处理是 CPU 最根本的任务。

1.3 计算机体系结构

1.3.1 计算机体系结构的发展

1. 计算机系统结构概述

计算机系统结构又称为计算机体系结构，也是计算机的属性及功能特征，即计算机的

外部特性。尽管不同的使用者所了解的计算机的属性有所不同,就通用计算机系统来说,计算机系统结构的属性应包括如下一些侧面:

- 硬件所能处理的数据类型;
- 所能支持的寻址方式;
- CPU 的内部寄存器;
- CPU 的指令系统;
- 主存的组织与主存的管理;
- 中断系统的功能;
- 输入输出设备及连接接口;
- 计算机体系结构类型。

2. 计算机体系结构分类

1) Flynn 分类法

1966 年 Flynn 提出了如下定义:

指令流(instruction stream)指机器执行的指令序列。

数据流(data stream)指由指令流调用的数据序列,包括输入数据和中间结果。

多倍性(multiplicity)指在系统最受限制的元件上同时处于同一执行阶段的指令或数据的最大可能个数。

按指令流和数据流的不同组织方式,把计算机体系结构分为如下 4 类:单指令流单数据流(SISD);单指令流多数据流(SIMD);多指令流单数据流(MISD);多指令流多数据流(MIMD)。

2) 冯式分类法

1972 年美籍华人冯泽云教授提出用最大并行度来对计算机体系结构进行分类。所谓最大并行度 P_m 是指计算机系统在单位时间内能够处理的最大的二进制位数。设每一个时钟周期 Δt_i 内能处理的二进制位数为 P_i , 则 T 个时钟周期内平均并行度为: $P_a = (\sum P_i)/T$, 其中 i 为 $1, 2, \dots, T$ 。平均并行度取决于系统的运行程度,与应用程序无关。所以系统在周期 T 内的平均利用率为 $\mu = P_a/P_m = (\sum P_i)/(T * P_m)$ 。

图 1-8 所示为用最大并行度对计算机体系结构的分类。用平面直角坐标系中的一点表示一个计算机系统,横坐标表示字宽(N 位),即在一个字中同时处理的二进制位数;纵坐标表示位片宽度(M 位),即在一个位片中能同时处理的字数,则最大并行度 $P_m = N * M$ 。由此得出 4 种不同的计算机结构:

- (1) 字串行、位串行(简称 WSBS), 其中 $N=1, M=1$;
- (2) 字并行、位串行(简称 WPBS), 其中 $N=1, M>1$;

(3) 字串行、位并行(简称 WSBP),其中 $N>1, M=1$;

(4) 字并行、位并行(简称 WPBP),其中 $N>1, M>1$ 。

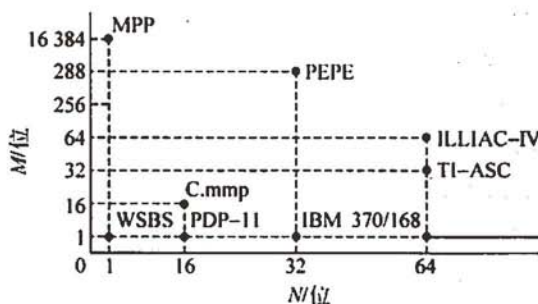


图 1-8 冯氏分类法

3. 计算机系统结构与计算机组成的区别

计算机系统结构所解决的问题是计算机系统在总体上、功能上需要解决的问题,而计算机组成要解决的是逻辑上如何具体实现的问题。

比如说指令系统的确定属于计算机系统结构的范畴,而指令的具体实现则属于计算机组成的内容。指令系统中要不要设置乘、除法指令是计算机系统结构要解决的问题,而一旦决定设置,具体用什么方法来实现乘、除法指令就属于计算机组成应解决的问题。

主存容量及编址方式的确定属于计算机系统结构,而具体如何构成主存则属于计算机组成。

可以想像,即使相同系统结构的计算机,但具体按此系统结构构成的计算机在实现方法、性能及价格上仍有很大差别。

4. 系统结构中并行性的发展

1) 并行性

并行性包括两个方面:同时性和并发性。

同时性指两个或两个以上的事件在同一时刻发生。

并发性指两个或两个以上的事件在同一时间间隔内连续发生。

充分利用并行性实现计算机的并行处理,可以提高计算机的处理速度。

2) 分类

从计算机信息处理的步骤和阶段的角度看,并行处理可分为:

- 存储器操作并行;
- 处理器操作步骤并行(流水线处理机);
- 处理器操作并行(阵列处理机);

- 指令、任务、作业并行(多处理机、分布处理系统、计算机网络)。

3) 并行性的发展

从 20 世纪 80 年代开始,在计算机系统结构上有了很大发展,相继出现了精减指令集计算机(RISC)、指令级上并行的超标量处理机、超级流水线处理机、超长指令计算机、多微处理机系统、数据流计算机等。

20 世纪 90 年代以来,最主要的发展是大规模并行处理(MPP),其中多处理机系统和多计算机系统是研究开发的热点。

1.3.2 存储系统

1. 存储器的层次结构

存储体系结构包括不同层次上的存储器,通过适当的硬件、软件有机地组合在一起形成计算机的存储体系结构。现在大多数人都将高性能计算机的存储体系结构描述成如图 1-9 所示的 3 层存储器层次结构。

三层存储结构是高速缓存(cache)、主存储器(MM)和辅助存储器(外存储器)。也有人将存储器层次分为 4 层,即将 CPU 内部的寄存器也看做存储器的一个层次。

有一些简单的计算机没有高速缓存,这样,计算机的存储体系就剩下主存和辅存两个层次。

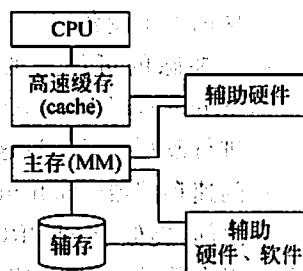


图 1-9 存储器层次结构示意图

2. 存储器的分类

1) 按位置分

按存储器所处的位置的不同,可分为内存和外存。

(1) 内存:也称为主存,设在主机内或主机板上,用来存放机器当前运行所需要的程序和数据,以便向 CPU 提供信息。相对于外存,其特点是容量小,速度快。

(2) 外存:也称为辅存,如磁盘、磁带、光盘等,用来存放当前不参加运行的大量信息,在需要时,可把需要的信息调入内存。相对于内存,外存的容量大,速度慢。

2) 按材料分

按构成存储器的材料,可分为磁存储器、半导体存储器和光存储器。

(1) 磁存储器:是用磁性介质做成的,如磁芯、磁泡、磁膜、磁鼓、磁带及磁盘等。

(2) 半导体存储器:根据所用元件又可分为双极型和 MOS 型;根据数据是否需要刷新,又可分为静态(static memory)和动态(dynamic memory)两类。

(3) 光存储器:如光盘(optical disk)存储器。

3) 按工作方式分

按工作方式可分为读写存储器和只读存储器。

(1) 读写存储器：既能读取数据也能存入数据的存储器。

(2) 只读存储器：根据数据的写入方式，这种存储器又可细分为 ROM、PROM、EPROM、EEPROM 等类型。

- 固定只读存储器(read only memory, ROM)：这种存储器是在厂家生产时就写好数据的，其内容只能读出，不能改变。一般用于存放系统程序 BIOS 和用于微程序控制。
- 可编程的只读存储器(programmable read only memory, PROM)：其中的内容可以由用户一次性地写入，写入后不能再修改。
- 可擦除可编程的只读存储器(erasable programmable read only memory, EPROM)：其中的内容既可以读出，也可以由用户写入，写入后还可以修改。改写的方法是，写入之前先用紫外线照射 15~20 分钟以擦除所有信息，然后再用特殊的电子设备写入信息。
- 电擦除的可编程的只读存储器(electrically erasable programmable read only memory, EEPROM)：与 EPROM 相似，EEPROM 中的内容既可以读出，也可以进行改写。只不过这种存储器用电擦除的方法进行数据的改写。
- 闪速存储器(flash memory)：简称闪存，闪存的特性介于 EPROM 和 EEPROM 之间，类似于 EEPROM，闪存也可使用电信号进行信息的擦除操作。整块闪存可以在数秒内删除，速度远快于 EPROM。

4) 按访问方式分

按访问方式可分为按地址访问的存储器和按内容访问的存储器。

5) 按寻址方式分

按寻址方式可分为随机存储器、顺序存储器和直接存储器。

(1) 随机存储器(random access memory, RAM)：这种存储器可对任何存储单元存入或读取数据，访问任何一个存储单元所需的时间是相同的。

(2) 顺序存储器(sequentially addressed memory, SAM)：访问数据所需要的时间与数据所在的存储位置相关，磁带是典型的顺序存储器。

(3) 直接存储器(direct addressed memory, DAM)：采用介于随机存取和顺序存取之间的一种寻址方式。磁盘是一种直接存取存储器，它对磁道的寻址是随机的，而在一个磁道内，则是顺序寻址。

3. 相连存储器

相连存储器是一种按内容访问的存储器。其工作原理就是把数据或数据的某一部分作为关键字，将该关键字与存储器中的每一单元进行比较，找出存储器中所有与关键字相同的数据字。相连存储器的结构如图 1-10 所示，各部件的功能如表 1-2 所示。

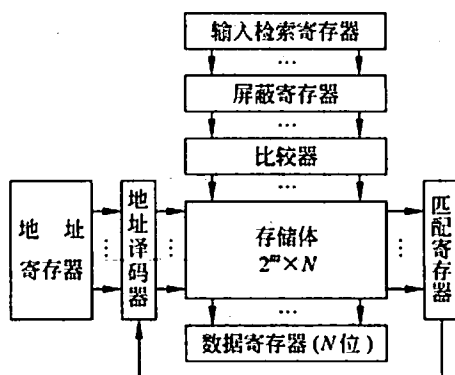


图 1-10 相连存储器的结构框图

表 1-2 部件功能说明

部 件	功 能
输入检索寄存器	用来存放要检索的内容(关键字)
屏蔽寄存器	用来屏蔽那些不参与检索的字段
比较器	将检索的关键字与存储体的每一单元进行比较。为了提高速度,比较器的数量应很大。对于位比较器,应每位对应一个,应有 $2^m \times N$ 个,对于字比较器应有 2^m 个
存储体	用于存放信息
匹配寄存器	用来记录比较的结果。它应有 2^m 个二进制位,用来记录 2^m 个比较器的结果,1 为相等(匹配),0 为不相等(不匹配)
数据寄存器	用来存放存储体中某个单元的内容
地址寄存器、地址译码器	使相连存储器具有按地址查找的功能

相连存储器可用在高速缓冲存储器中;在虚拟存储器中用来作段表、页表或快表存储器;用在数据库和知识库中。

4. 高速缓存

高速缓存是用来存放当前最活跃的程序和数据的,作为主存局部域的副本,其特点是:容量一般在几 KB 到几 MB 之间;速度一般比主存快 5 到 10 倍,由快速半导体存储器构成;其内容是主存局部域的副本,对程序员来说是透明的。

1) 高速缓存的组成

高速缓存的组成如图 1-11 所示。由图 1-11 可以看到,cache 由两部分组成:控制部分和 cache 部分。

cache 部分用来存放主存的部分拷贝(副本)信息。控制部分的功能是:判断 CPU 要

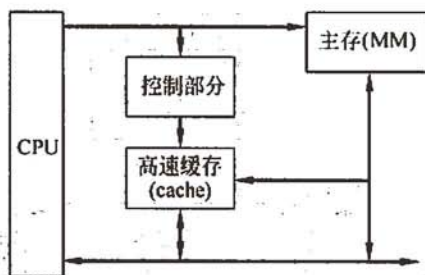


图 1-11 高速缓存的构成框图

访问的信息是否在 cache 中,若在即为命中,若不在则没有命中。命中时直接对 cache 存储器寻址。未命中时,要按照替换原则,决定主存的一块信息放到 cache 的哪一块里面。

2) 高速缓存中的地址映像方法

在 CPU 工作时,送出的是主存的地址,而应从 cache 中读写信息。这就需要将主存地址转换成 cache 的地址,这种地址的转换叫做地址映像,cache 的地址映像有三种方法。

(1) 直接映像:指主存的块与 cache 中块的对应关系是固定的,如图 1-12 所示。

在这种映像方式下,由于主存中的块只能存放在 cache 的相同块号中,因此,只要主存地址中的主存区号与 cache 中的主存区号相同,则表明访问 cache 命中。一旦命中,按主存地址中的区内块号立即可得到要访问的 cache 中的块,而块内地址就是主存地址中给出的低位地址,见图 1-12。

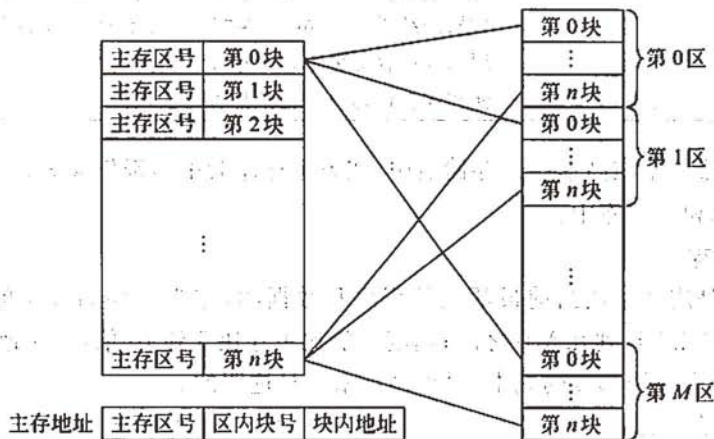


图 1-12 直接映像示意图

直接映像这种方式的优点是地址变换很简单,只要主存地址中主存区号找到了,则按

主存地址中后面的区内块号和块内地址立刻就找到了该块及块内的存储单元。其缺点是灵活性差,例如不同区号中块号相同的块无法同时调入 cache,即使 cache 中有空着的块也只能空着。

(2) 全相连映像: 全相连映像的示意图如图 1-13 所示。

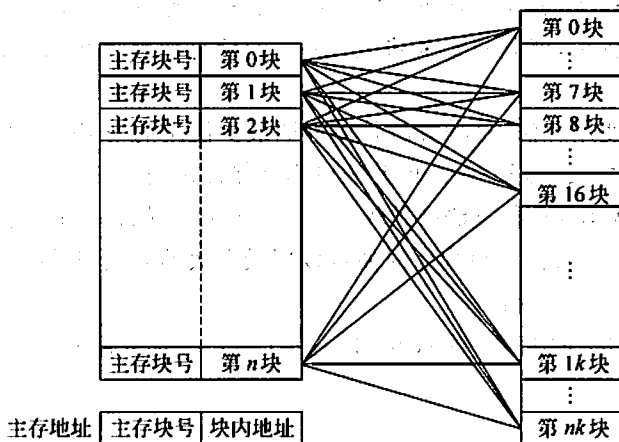


图 1-13 全相连映像示意图

同样,主存与 cache 存储器均分成容量相同的块,这种映像方式允许主存的任一块可以调入 cache 存储器的任何一个块的空间中。

在地址变换时,利用主存地址高位表示的主存块号与 cache 中的主存块号进行比较,若相同即为命中。这时可根据块号所对应的块就知道要访问的是哪一块。cache 的块找到后,块内地址就是主存的低位地址。这样便可以读写 cache 块中的内容。在变换时当找到主存块号命中时,还必须知道主存的这一块存到了 cache 的哪一块里面。

全相连映像的主要优点是主存的块调入 cache 的位置不受限制,十分灵活。其主要缺点是无法从主存块号中直接获得 cache 的块号,变换比较复杂,速度比较慢。

(3) 组相连映像: 这种方式是前面两种方式的折中。具体做法是将 cache 中的块再分成组。例如,假定 cache 有 16 块,再将每两块分为 1 组,则 cache 就分为 8 组。主存同样分区,每区 16 块,再将每两块分为 1 组,则每区就分为 8 组。

组相连映像就是规定组采用直接映像方式而块采用全相连映像方式。也就是说主存任何区的 0 组只能存到 cache 的 0 组中,1 组只能存 1 组,依次类推。组内的块则采用全相连映像方式,即一组内的块可以任意存放。也就是说主存一组中的任一块可以存入 cache 相应组中任一块中。

这种方式下,通过直接映像方式来决定组号,在一组内再用全相连映像方式来决定

cache 中的块号。由主存地址高位决定主存区号与 cache 中区号比较可决定是否命中。主存后面的地址即为组号。但组块号要根据全相连映像方式由记录可以决定组内块号。

3) 替换算法

替换算法的目标就是使 cache 获得最高的命中率。常用算法有：

(1) 随机替换算法,就是用随机数发生器产生一个要替换的块号,将该块替换出去。

(2) 先进先出算法,就是将最先进入 cache 的信息块替换出去。此法简单但最先进入的并非不经常用。

(3) 近期最少使用算法,这种方法是将近期最少使用的 cache 中的信息块替换出去。该算法较先进先出算法要好一些。但此法也不能保证过去不常用将来也不常用。

(4) 优化替换算法,这种方法必须先执行一次程序,统计 cache 的替换情况。有了这样的先验信息,在第二次执行该程序时便可以用最有效的方式来替换,达到最优的目的。

4) 高速缓存的性能分析

若 H 为 cache 的命中率; t_c 为 cache 的存取时间; t_m 为主存的访问时间。则 cache 的等效访问时间 t_a 为:

$$t_a = H t_c + (1 - H) t_m$$

使用 cache 比不使用 cache 的 CPU 访问存储器速度提高的倍数 r 可用下式求得:

$$r = t_m / t_a$$

5. 虚拟存储器

虚拟存储器是由主存、辅存、存储管理单元及操作系统中存储管理软件组成的存储系统。在程序员使用该存储系统时,可以使用的内存空间可以远远大于主存的物理空间。但实际上并不存在那么大的主存,故称其为虚拟存储器。虚拟存储器分为以下几类。

页式虚拟存储器是以页为信息传送单位的虚拟存储器。通常一页为几百字节到几千字节。实现页式管理,须建立虚页与实页间的关系表,称为页表;在页表及变换软件的控制下,可将程序的虚拟地址变换为主存的实地址。页式管理的优点是:页表硬件少,查表速度快;主存零头少。其缺点是:分页无逻辑意义,不利于存储保护。

段式虚拟存储器是以程序的逻辑结构形成的段(如某一独立程序模块、子程序等)作为主存分配依据的一种段式虚拟存储器的管理方法。为实现段式管理,需建立段表;在段地址变换机构及软件的控制下,可将程序的虚拟地址变换为主存的实地址。段式管理的优点是:段的界限分明;支持程序的模块化设计;易于对程序段的编译、修改和保护;便于多道程序的共享。主要缺点:因段的长度不一,主存利用率不高,产生大量内存碎片,造成浪费;段表庞大,查表速度慢。

段页式虚拟存储器是页式虚拟存储器和段式虚拟存储器两者相结合的一种管理方

式。在这种虚拟存储器中,程序按逻辑结构分段,每一段再分成若干大小固定的页。程序的调入调出是按页进行的,而程序又可按段实现保护。因此,这种管理方式兼有前两者的优点,只是地址变换速度比较慢。

从以上的描述可以看到,虚拟存储器将大容量的外存也纳入存储器的管理范围。但在具体执行程序时需判断程序是否在内存中,若不在(可认为不命中),则需从辅存中调入。这种思路与前面描述的 cache 中的替换一样。因此,虚拟存储器中的替换算法与前面所述的一样,此处不再说明。

6. 外存储器

外存储器用来存放暂时不用的程序和数据,并且以文件的形式存储。CPU 不能直接访问外存中的程序和数据,只有将其以文件为单位调入主存方可访问。外存储器由磁表面存储器(如磁盘、磁带)及光盘存储器构成。下面介绍两种常用的外存储器。

1) 磁盘存储器

在磁表面存储器中,磁盘的存取速度较快,且具有较大的存储容量,故是目前广泛使用的外存储器。

(1) 磁盘存储器的构成:磁盘存储器由盘片、驱动器、控制器和接口组成。盘片用来存储信息;驱动器用于驱动磁头沿盘面径向运动以寻找目标磁道位置,驱动盘片以额定速率稳定旋转,并且控制数据的写入和读出。控制器接受主机发来的命令,将它转换成磁盘驱动器的控制命令,并实现主机和驱动器之间数据格式的转换及数据传送,以控制驱动器的读写操作。一个控制器可以控制一台或多台驱动器。接口是主机和磁盘存储器之间的连接逻辑。在微机系统中,通常将磁盘控制器与接口制作在一块插件上,称为磁盘适配卡。

另外,在软盘驱动器中,盘片可以随便拆卸,因而盘片可与驱动器分离。但在多数硬盘存储器中,盘片常密封于驱动器之中,不可分离。

(2) 磁盘存储器的种类:磁盘存储器有两种,一种是以软质聚酯塑料薄片为基体,在基体上涂敷氧化铁磁性材料作为记录介质,称为软盘;另一种是采用硬质基体,在基体上生成一层很薄但很均匀的记录磁层,称为硬盘。下面分别介绍软盘和硬盘。

- 软盘:盘片的直径和记录密度可能不同,目前使用的是 3.5 英寸高密度软盘。软盘的盘面封装在一个保护套内,保护套内有一层无纺布,用来防尘,消除静电,保护盘片表面以免划伤。保护套中心的大圆孔是装卡孔,起定位作用。保护套上还有一个读写口、写保护口及索引孔。盘片旋转时磁头通过读写口对盘片进行读写操作。若要防止软盘中的文件不被修改和删除,应让软盘处于写保护。

为了正确存储信息,将盘片划成许多同心圆,称为磁道,从外到里编号,最外一圈为 0 道,往内道号依次增加。沿径向的单位距离的磁道数称为道密度,单位为 tpi(每英寸磁

道数)。将一个磁道沿圆周等分为若干段,每段称为一个扇段或扇区,每个扇区内可存放一个固定长度的数据块,如 512B。磁道上单位距离可记录的比特数称为位密度,单位为 bpi(每英寸比特数)。因为每条磁道上的扇区数相同,而每个扇区的大小又一样,所以每条磁道都记录同样多的信息。又因为里圈磁道圆周比外圈磁道的圆周小,所以里圈磁道的位密度要比外圈磁道的位密度高。最内圈的位密度称为最大位密度。

访问软盘时应给出软盘驱动器号(A 盘或 B 盘)、磁头号、磁道号、扇区号、交换量(指文件占有扇区数)等寻址信息。

软盘的记录格式是指磁盘表面上信息的存储格式。为了盘结构的互换性和简化系统设计,采用统一的标准记录格式,ISO 已确定将 IBM 记录格式作为国际标准。图 1-14 为 IBM34 系统磁道格式。

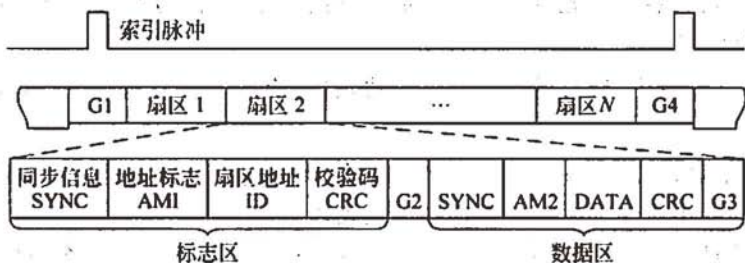


图 1-14 IBM34 系统磁道格式

有的操作系统允许用户使用非标准记录格式,以保护自己软件的产权,防止非法复制。

一般新买来的软盘,在使用前要用磁盘命令进行格式化。格式化分为物理格式化(低级格式化)和逻辑格式化(高级格式化)。物理格式化把磁道划分为若干扇区,每个扇区又划分为标志区和数据区,并将有关信息写入磁盘,但 DATA 段空着。逻辑格式化完成:建立文件目录表、磁盘扇区分配表、磁盘参数表等。

实际上有两种磁盘容量指标。一种是非格式化容量,它是指一个磁盘所能存储的总位数;另一种是格式化容量,它是指各扇区中 DATA 区容量总和。计算公式分别如下:

非格式化容量 = 面数 × (磁道数/面) × 内圆周长 × 最大位密度

格式化容量 = 面数 × (磁道数/面) × (扇区数/道) × (字节数/扇区)

- 硬盘: 按盘片是否固定、磁头是否移动等指标硬盘可分为移动磁头固定盘片的磁盘存储器、固定磁头的磁盘存储器、移动磁头可换盘片的磁盘存储器和温彻斯特磁盘存储器(简称温盘)。按盘片的直径,分为 14 英寸、8 英寸、5.25 英寸、3.5 英寸几种。

一个硬盘驱动器内可装多个盘片,组成盘片组,每个盘片都配有一个独立的磁头。所有记录面上相同序号的磁道构成一个圆柱面,其编号与磁道编号相同。文件存储在硬盘上时尽可能放在同一圆柱面上,或者放在相邻柱面上,这样可以缩短寻道时间。

硬盘的寻址信息由硬盘驱动号、圆柱面号、磁头号(记录面号)、数据块号(或扇区号)以及交换量组成。软盘和硬盘在结构和性能上的区别如下:

- 硬盘转速高,存取速度快;软盘转速低,存取速度慢;
- 硬盘有固定头、固定盘、盘组等结构;软盘都是活动头、可换盘片结构;
- 硬盘是浮动磁头读写,磁头不接触盘片;软盘磁头是接触式读写;
- 硬盘系统价格较贵;软盘价格低,盘片保存使用简便。

2) 光盘存储器

光盘存储器是一种采用聚焦激光束在盘式介质上非接触地记录高密度信息的新型存储装置。

(1) 光盘存储器的类型:根据性能和用途,可分为只读型光盘(CD-ROM)、只写一次型光盘(WORM)和可擦除型光盘。只读型光盘是由生产厂家预先用激光在盘片上蚀刻不能再改写的各种信息,目前这类光盘使用很普遍。只写一次型光盘是指由用户一次写入、可多次读出但不能擦除的光盘。写入方法是利用聚焦激光束的热能,使光盘表面发生永久性变化而实现的。可擦除型光盘是读写性光盘,它利用激光照射引起介质的可逆性物理变化来记录信息。

(2) 光盘存储器的组成及特点:光盘存储器由光学、电学和机械部件等组成。其特点是记录密度高;存储容量大;采用非接触式读写信息(光头距离光盘通常为 2mm);信息可长期保存(其寿命达 10 年以上);采用多通道记录时数据传送率可超过 200MB/s;制造成本低;对机械结构的精度要求不高;存取时间较长。

光盘存储器与磁盘存储器的比较如下:

- 光盘是非接触式读写信息,比磁盘的头盘间距大 1 万倍左右,所以光盘的耐用性高,使用寿命长。
- 光盘可靠性高,对使用环境要求不高,机械振动上的问题较少,不需要特殊的防震与除尘设备。
- 光盘的记录密度为磁盘的 10~100 倍,但取数时间慢于磁盘,其读写速度只有磁盘的几分之一。
- 光盘易于更换,可做成自动换盘装置。

7. 磁盘阵列技术

磁盘阵列是由多台磁盘存储器组成的、一个快速大容量高可靠的外存子系统。现在常见的称为廉价冗余磁盘阵列(RAID)。目前,RAID 分为 6 级,如表 1-3 所示。

表 1-3 廉价冗余磁盘阵列

RAID 级别	说 明
RAID0	0 级廉价冗余磁盘阵列是一种不具备容错能力的阵列。由 N 个磁盘存储器组成的 0 级阵列,其平均故障间隔时间(MTBF)是单个磁盘存储器的 N 分之一,但数据传输率是单个磁盘存储器的 N 倍
RAID1	1 级廉价冗余磁盘阵列是采用镜像容错技术改善可靠性的一种磁盘阵列
RAID2	2 级廉价冗余磁盘阵列是采用汉明码作错误检测的一种磁盘阵列
RAID3	3 级廉价冗余磁盘阵列这种磁盘阵列减少了用于检验的磁盘存储器的台数,从而提高了磁盘阵列的有效容量。一般只有一个检验盘
RAID4	4 级廉价冗余磁盘阵列是一种可独立地对组内各磁盘进行读写的磁盘阵列,该阵列也只用一个检验盘
RAID5	5 级廉价冗余磁盘阵列该阵列是对 RAID4 的一种改进,它不设置专门的检验盘。同一台磁盘上既记录数据,也记录检验信息。这就解决了前面多台磁盘机争用一台检验盘的问题

除此之外,目前还有 RAID6、RAID7、RAID10 等,它们均是对前者的改进,此处不再说明。

1.3.3 CISC/RISC

1. 指令系统的发展

CISC 的含义是复杂指令集计算机。众所周知,早期的计算机指令系统比较简单,通常只有十几到几十条指令。随着计算机的发展,指令系统的也随之发展,不仅指令条数增加,而且指令的功能也有了许多增加。其目的是利用增加指令功能和复杂程度缩小汇编与高级语言的差距。另外,为了使新老型号的系列 CPU 的指令系统向上兼容,就需要在新的 CPU 中既要保持老的指令系统,又要增加新的指令。例如,80x86 系列 CPU,每更新一次,必然增加一些指令。

由于上述原因使得 CPU 的指令系统越来越庞大,越来越复杂。这就是复杂指令集计算机(CISC)。由于指令系统的复杂性,使 CPU 硬件也变得十分复杂,同时也限制了 CPU 的运行速度。因此,复杂指令集计算机(CISC)在性能上的提高遇到了很大的困难。

2. 精简指令集计算机(RISC)

人们对典型的 CISC 执行程序中指令使用频度进行统计发现,指令系统中只有大约 20%的指令被经常使用,其使用频度达 80%,而且这些指令都是一些加、传送、转移等最简单的指令。也就是说大多数的复杂指令只有 20%的使用概率。

若只保留 20%的最简单的指令,使指令尽可能简单,从而设计一种硬件结构十分简单、执行速度很高的 CPU。这就是精简指令集计算机(RISC)。

3. RISC 的特点

RISC 简化了 CPU 的控制器,同时提高了处理速度,它的特点如下:

- (1) 指令种类少。一般只有十几到几十条简单的指令。
- (2) 指令长度固定,指令格式少。这可使指令译码更加简单。
- (3) 寻址方式少。适合于组合逻辑控制器,便于提高速度。
- (4) 设置最少的访内指令。访问内存比较花时间,尽量少用。
- (5) 在 CPU 内部设置大量的寄存器,使大多数操作在速度很快的 CPU 内部进行。
- (6) 非常适合流水线操作。由于指令简单,并行执行就更易实现。

1.3.4 输入输出技术

1. 微型计算机中最常用的内存与接口的编址方式

尽管在微型计算机中存在着许多种内存与接口地址的编址方法,但最常见到的是下面将要描述的两种,而且理解了这两种编址方法,再遇到其他编址方法也就不难理解了。

1) 内存与接口地址独立的编址方法

这种编址方法也有人称为内存与接口地址隔离的编址方法。这种方法就是,在微型机中,内存地址和接口地址是完全独立的两个地址空间,它们是完全独立的并且是相互隔离的。

在使用中,这种编址方法的地址很清楚,内存就用于存放程序和数据,而接口就用于寻址外设。所使用的指令也完全不同,用于接口的指令只用于接口读写,其余的指令全都是用于内存的。因此,在编程序或读程序中很易使用和辨认。

这种编址方法的缺点就是用于接口的指令太少,功能太弱。

2) 内存与接口地址统一的编址方法

这种编址方法也有人称为内存与接口地址混合的编址方法。在这种编址方法里,内存地址和接口地址统一在一个公共的地址空间里。也就是说内存和接口共用这些地址。在这些地址空间里拿出某一些地址分配给接口使用而剩下的就可以归内存所用。也就是说,地址空间里的每一个地址都可以分配给接口也可以分配给内存使用。但是,分配给内存的只能用于内存,接口绝不允许使用。同样,分配给接口的地址内存也绝不能再使用。

这种编址方法的优点是原则上用于内存的指令全都可以用于接口。这就大大地增强了对接口的操作功能。而且在指令上也不再区分内存或接口指令,也就是说两者用的指令全都是一样的。

该编址方法的缺点就在于整个地址空间被分成两部分。其中一部分分配给接口使用,剩余的为内存所用,这经常会导致内存地址不连续。再就是用于内存的指令和用于接口的指令是完全一样的,这在读程序时就需根据参数定义表仔细加以辨认。

2. 直接程序控制

在完成外设数据的输入输出中,整个输入输出过程是在 CPU 执行程序的控制下完成的。这种方式分为以下两种情况。

1) 无条件传送

在此情况下,外设总是准备好的,它可以无条件地随时接收 CPU 发来的输出数据,也能够无条件地随时向 CPU 提供需要输入的数据。

2) 程序查询方式

在这种方式下,利用查询方式进行输入输出,就是通过 CPU 通过执行程序查询外设的状态,判断外设是否准备好接收数据或准备好了向 CPU 输入的数据。根据这种状态,CPU 有针对性地为外设的输入输出服务。

通常,一个计算机系统中可以存在着多种不同的外设,如果这些外设用查询方式工作,则 CPU 应对这些外设逐一进行查询,发现哪个外设准备就绪就对该外设服务。其过程如图 1-15 所示。

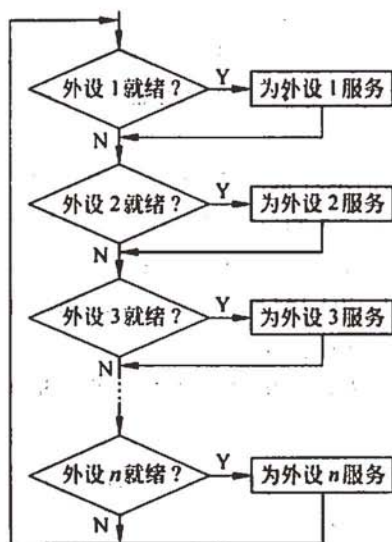


图 1-15 多台外设查询工作

上面对查询方式的描述可以看到,这种工作方式有两大缺点:

(1) 降低了 CPU 的效率。在这种工作方式下,CPU 不做别的事,只是不停地对外设的状态进行查询。在实际的工程应用中,对那些慢速的外设在不影响到外设工作时,CPU 可以抽空做一些别的事。

(2) 对外部的突发事件无法做出实时响应。

查询方式的优点就在于这种思想很易理解,同时实现这种方式工作也很容易。

3. 中断方式

由程序控制 I/O 的方法其主要缺点在于 CPU 必须等待 I/O 系统完成数据传输任务,在此期间 CPU 需定期地查询 I/O 系统的状态,以确认传输是否完成。因此整个系统的性能严重下降。

为了克服该缺陷,于是把中断机制引入到 I/O 传输过程中。CPU 利用中断方式完成数据的输入输出:当 I/O 系统与外设交换数据时,CPU 无需等待也不必去查询 I/O 的状态,而可以抽身出来处理其他任务。当 I/O 系统完成了数据传输后则以中断信号通知 CPU。CPU 然后保存正在执行程序的现场,转入 I/O 中断服务程序,完成与 I/O 系统的数据交换,然后再返回原主程序继续执行。与程序控制方式相比,中断方式因为 CPU 无须等待而提高了效率。中断方式的输入与输出过程如图 1-16 所示。

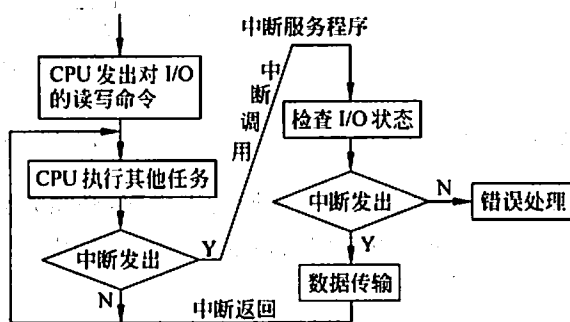


图 1-16 中断方式输入输出

1) 中断处理方法

在系统中具有多个中断源的情况下,常用的处理方法有:多中断信号线法(multiple interrupt lines)、中断软件查询法(software poll)、雏菊链法(daisy chain)、总线仲裁法和中断向量表法。

(1) 多中断信号线法:每个中断源都有属于自己的一根中断请求信号线向 CPU 提出中断请求。

(2) 中断软件查询法:当 CPU 检测到一个中断请求信号以后,即转入到中断服务程序去轮询每个中断源以确定是谁发出了中断请求信号。对各个设备的响应优先级由软件设定,如图 1-17 所示。

(3) 雏菊链法:软件查询的缺陷在于花费时间太多。雏菊链法实际上是一种硬件查询法。所有的 I/O 模块共享一根共同的中断请求线。而中断确认信号则以链式在各模块间相连。当 CPU 检测到中断请求信号,则发出中断确认信号。中断确认信号依次在

I/O 模块间传递,直到发出请求的模块,该模块则把它的 ID 送往数据线由 CPU 读取,见图 1-18。

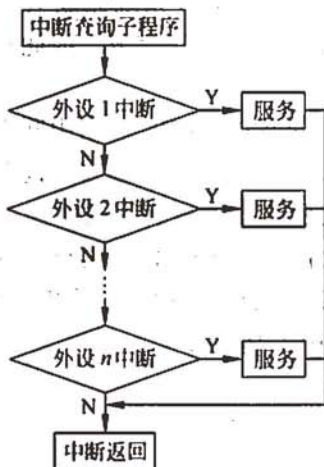


图 1-17 中断软件查询法

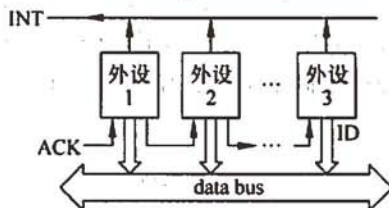


图 1-18 菊花链法

(4) 总线仲裁法：一个 I/O 设备在发出中断请求之前,必须先获得总线控制权。所以可由总线仲裁机制来裁定谁可以发出中断请求信号。当 CPU 发出中断响应信号后,该设备即把自己 ID 发往数据线。

(5) 中断向量表法：中断向量表用来保存各个中断源的中断服务程序的入口地址。当外设发出中断请求信号（INTR）以后,由中断控制器（INTC）确定其中断号,并根据中断号查找中断向量表来取得其中断服务程序的入口地址;同时,INTC 把中断请求信号提交给 CPU,如图 1-19 所示。中断源的优先级由 INTC 来控制。

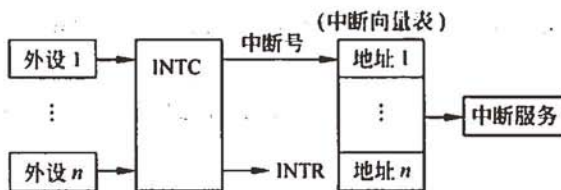


图 1-19 中断向量表法

2) 中断优先级控制

在具有多个中断源的计算机系统中,这些中断源对服务的要求紧迫程度可能不同。在这样的计算机系统中就需要按中断源的轻重缓急来安排对它们的服务。

在中断优先级控制系统中给最紧迫的中断源分配高的优先级而给那些要求相对不紧迫(例如几百微秒到几毫秒)的中断源分配低一些的优先级。在进行优先级控制时解决以下两种情况。

(1) 当不同优先级的多个中断源同时提出中断请求时,CPU 应优先响应优先级最高的中断源。

(2) 当 CPU 正在对某一个中断源服务时,又有比它优先级更高的中断源提出中断请求,CPU 应能暂时中断正在执行的中断服务程序而转去对优先级更高的中断源服务,服务结束后再回到原先被中断的优先级较低的中断服务程序继续执行。这种情况称为中断嵌套,即中断服务程序中嵌套着另一个中断服务程序。

4. 直接存储器存取(DMA)方式

在计算机与外设交换数据的过程中,无论无条件传送、利用查询方式传送,还是利用中断方式传送,都需要由 CPU 通过执行程序来实现。这就限制了数据的传送速度。

直接内存存取(direct memory access,DMA),是指数据在内存与 I/O 设备间的直接成块传送。即在内存与 I/O 设备间传送一个数据块的过程中,不需要 CPU 的任何干涉,只需要 CPU 在过程开始启动(即向设备发出“传送一块数据”的命令)与过程结束(CPU 通过轮询或中断得知过程是否结束和下次操作是否准备就绪)时的处理,实际操作由 DMA 硬件直接执行完成,CPU 在此传送过程中可做别的事情。

DMA 传送的一般过程如图 1-20 所示。

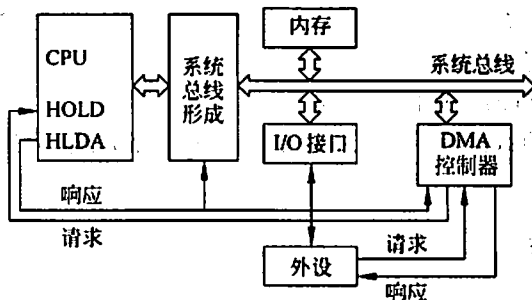


图 1-20 DMA 过程示意图

- ① 外设向 DMA 控制器(DMAC)提出 DMA 传送的请求。
- ② DMA 控制器向 CPU 提出请求,其请求信号通常加到 CPU 的保持请求输入端 HOLD 上。
- ③ CPU 在完成当前的总线周期后立即对此请求做出响应。CPU 的响应包括两个方面的内容:一方面 CPU 将有效的保持响应信号 HLDA 输出加到 DMAC 上,告诉 DMAC

它的请求已得到响应;同时,另一方面 CPU 将其输出的总线信号置为高阻,这就意味着 CPU 放弃了对总线的控制权。

④ 此时,DMAC 获得了对系统总线的控制权,开始实施对系统总线的控制。同时向提出请求的外设送出 DMAC 的响应信号,告诉外设其请求已得到响应,现在准备开始进行数据的传送。

⑤ DMAC 送出地址信号和控制信号,实现数据的高速传送。

⑥ 当 DMAC 将规定的字节数传送完时,它就将 HOLD 信号变为无效并加到 CPU 上。撤销对 CPU 的请求。CPU 检测到无效的 HOLD 就知道 DMAC 已传送结束,CPU 就送出无效的 HLDA 响应信号,同时重新获得系统总线的控制权,接着 DMA 前的总线周期继续执行下面的总线周期。

在此再强调说明,在 DMA 传送过程中无须 CPU 干预,整个系统总线完全交给了 DMAC,由它控制系统总线完成数据传送。

5. 输入输出处理机(IOP)

上面所描述的输入输出方式适合于外设不多,速度不很高的小型或微型机中,实现起来不是很复杂。在大型计算机中,外设很多,要求计算机的速度很高。采用程序传送、查询、中断或 DMA 均会因输入输出而造成过大的开销,影响计算机的整体性能。为此,提出采用输入输出处理机。

1) 输入输出处理机的功能

输入输出处理机是一个专用处理机,接在主计算机上,主机的输入输出操作由它来完成。它根据主机的 I/O 命令,完成对外设数据的输入输出。既然输入输出由 IOP 来完成了,主机的工作效率必然提高了。

2) 输入输出处理机的数据传送方式

输入输出处理机的数据传送方式有 3 种:字节多路方式、选择传送方式和数组多路方式。

1.3.5 流水线操作

1. 指令流水线

指令流水线的概念就是将一条指令分解成一连串执行的子过程,在 CPU 中变一条指令的串行执行子过程为若干条指令的子过程在 CPU 中重叠执行,这就是指令流水线的思路。如果能做到每条指令均分解为 m 个子过程,且每个子过程的执行时间都一样,则利用此条流水线可将一条指令的执行时间由原来的 t 缩短为 t/m 。

1) 流水的基本概念

流水线技术是将一个重复的时序分解成若干个子过程,而每一个子过程都可有效地在其专用功能段上与其他子过程同时执行。流水线技术应用于计算机系统结构的各个方

面,在此以指令的执行过程为例介绍流水线技术。

前面已举例介绍过,可将指令的执行过程分解成取指令、分析指令和执行指令3个子过程。早期指令的执行是顺序方式,即现行指令执行完毕后才开始读取后继指令。这种处理方式控制简单,且比较直观,但在时间安排上不能充分利用各部件。为了提高工作速度,现在的大多数计算机都在不同程度上采取重叠处理方式,重叠的程度取决于存储体与运算部件的多少及控制指令部件的工作方式。

在一次重叠处理时,可将指令的执行过程粗分为分析和执行两个子过程。当第I条指令在指令部件中分析完毕后,将进入执行部件去实现指令的操作。此时指令分析部件处于“空闲”状态,若利用这个“空闲”状态对I+1条指令进行分析,使其与第I条指令的执行同步进行,也即两条指令在时间上存在着重叠,如图1-21所示。

若把指令的执行过程进一步细分为取指令、指令译码、取操作数和执行四个子过程,并改进运算器的结构以加快其执行子过程,则得到如图1-22所示的流水处理的时空图,其中的1、2、3、4、5表示要处理的5条指令。一次重叠与流水的区别在于前者将一条指令的执行分为两个子过程,而后者却分为4个或多个子过程。一次重叠可同时执行两条指令,而流水方式则可同时执行多条指令。



图 1-21 一次重叠处理

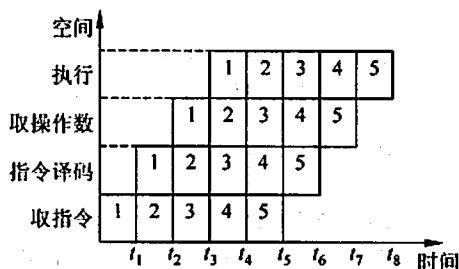


图 1-22 流水线处理的时空图

2) 流水线技术的特点

- (1) 流水线可分成若干个相互联系的子过程;
- (2) 实现子过程的功能所需时间尽可能相等;
- (3) 形成流水处理,需要一段准备时间;
- (4) 指令流发生不能顺序执行时,会使流水线过程中断,再形成流水线过程则需要时间。

3) 流水线结构的分类

流水线结构的类型众多,并且分类方法各异,常见的几种分类方法如下所示。

(1) 按完成的功能分类:

- 单功能流水线。只完成一种固定功能的流水线,如只能实现浮点加。

- 多功能流水线。同一流水线上可有多种连接方式来实现多种功能,如 ASC 运算器中的 8 个可并行工作的功能块,可按不同的连接方式实现浮点加、减或定点乘法运算。

(2) 按同一时间内各段之间的连接方式分类:

- 静态流水线。同一时间流水线上的所有功能块只能按同一种运算的连接方式工作。如 ASC 运算器中的 8 个可并行工作的功能块,或都按浮点加、减运算连接,或都按定点乘法运算连接。
- 动态流水线。同一时间流水线上的所有功能块可按不同种运算的连接方式工作。

(3) 按数据表示分类:

- 标量流水线处理机。只能对标量数据进行流水处理。
- 向量流水线处理机。它具有向量指令,可对向量的各元素进行流水处理。

2. 流水线处理机的主要指标

(1) 吞吐率。吞吐率是指单位时间里流水线处理机流出的结果数。对指令而言就是单位时间里执行的指令数。如果流水线的子过程所用时间不一样长,则吞吐率 p 应为最长子过程的倒数,即:

$$p = 1 / \max\{\Delta t_1, \Delta t_2, \dots, \Delta t_m\}$$

(2) 建立时间。流水线开始工作,须经过一定时间才能达到最大吞吐率,这就是建立时间。若 m 个子过程所用时间一样,均为 Δt_0 ,则建立时间 $T_0 = m\Delta t_0$ 。

1.3.6 总线结构

1. 总线的定义与分类

广义地讲,任何连接两个以上电子元器件的导线都可以称为总线。通常分为 4 类:

- (1) 芯片内总线,用于在集成电路芯片内部各部分的连接。
- (2) 元件级总线,用于一块电路板内各元器件的连接。
- (3) 内总线,又称系统总线,用于构成计算机各组成部分(CPU、内存、接口等)的连接。
- (4) 外总线,又称通信总线,用于计算机与外设或计算机与计算机的连接或通信。

2. 内总线

内总线有专用内总线和标准内总线。内总线的性能直接影响到计算机的性能。自计算机发明,尤其是微型机诞生以来,内总线的标准已超过百条。常见的内总线标准如下所述。

1) ISA 总线

ISA 是工业标准总线。它向上兼容更早的 PC 总线,在 PC 总线 62 个插座信号的基础上,再扩充另一个 36 个信号的插座构成 ISA 总线。

ISA 总线主要包括 24 个地址线,16 条数据线,控制总线(内存读写、接口读写、中断请求、中断响应、DMA 请求、DMA 响应等等), $\pm 5V$ 、 $\pm 12V$ 电源、地线等。

2) EISA 总线

该总线是在 ISA 总线的基础上发展起来的 32 位总线。该总线定义 32 位地址线,32 位数据线,以及其他控制信号线、电源线、地线等共 196 个接点。总线传输速率达 33MB/s。该总线利用总线插座与 ISA 总线相兼容,插板插在上层为 ISA 总线信号;将插板插到下层便是 EISA 总线。

3) PCI 总线

PCI 总线是目前微型机上广泛采用的内总线。PCI 总线有适于 32 位机的 124 个信号的标准和适于 64 位机的 188 个信号的标准。PCI 总线的传输速率至少为 133MB/s,64 位 PCI 总线的传输速率为 266MB/s,具有很高的传输速率。PCI 总线的工作与处理器的工作是相互独立的,也就是说 PCI 总线时钟与处理器时钟是独立的、非同步的。PCI 总线上设备是即插即用的。

3. 外总线

外总线的标准有七八十种之多,此处仅介绍下面几种。

1) RS-232C

RS-232C 是一条串行外总线,其主要特点是:所需传输线比较少,最少只需三条线(一条发、一条收、一条地线)即可实现全双工通信。传送距离远,用电平传送为 15 米,电流环传送可达千米。有多种可供选择的传送速率。采用非归零码负逻辑工作,电平 $\leq -3V$ 为逻辑 1,而电平 $\geq +3V$ 为逻辑 0,具有较好的抗干扰性。

2) SCSI 总线

小型计算机系统接口(SCSI)是一条并行外总线,广泛用于连接软硬磁盘、光盘、扫描仪等。该接口总线早期是 8 位的,后来发展到 16 位。传输速率由 SCSI-1 的 5MB/s 到 16 位的 Ultra2 SCSI 的 80MB/s。今天的传输速率已高达 320MB/s。该总线上最多可接 63 种外设。传道距离可达 20m(差分传送)。

3) USB

通用串行总线 USB 当前风头正劲,近几年得到十分广泛的应用。USB 由 4 条信号线组成,其中两条用于传送数据,另外两条传送 $+5V$ 容量为 500mA 的电源。可以经过集线器 HUB 进行树状连接,最多可达 5 层。该总线上可接 127 个设备。USB1.0 有两种传送速率:低速 1.5Mb/s,高速为 12Mb/s。USB2.0 的传送速率为 480Mb/s。USB 总线最大的优点还在于它支持即插即用技术并支持热插拔。

4) IEEE 1394

这是另一条串行外总线,近几年同样得到十分广泛的应用。IEEE 1394 由 6 条信号线组成,其中两条用于传送数据,两条传送控制信号,另外两条传送 8 到 40V 容量为

1500mA 的电源。资料上介绍,IEEE 1394 总线上可接 63 个设备。IEEE 1394 的传送速率从 400Mb/s,800Mb/s,1600Mb/s 直到 3.2Gb/s。而这种总线最大的优点也在于它支持即插即用并支持热插拔。

1.3.7 多处理机与并行处理

1. 阵列处理机

1) 阵列处理机的概念

在前面已经提到有关并行处理的概念。在这里专门介绍阵列处理机。阵列处理机又称并行处理机,它将重复设置的多个处理单元(PU)按一定方式连成阵列,在单个控制部件(CU)控制下,对分配给自己的数据进行处理,并行地完成一条指令所规定的操作。这是一种单指令流多数据流(SIMD)计算机,通过资源重复实现并行性,如图 1-23 所示。

2) SIMD 计算机的互连网络

SIMD 计算机的互连网络的设计目标:结构简单、灵活;处理单元间信息传送的步数尽可能少。

(1) 立方体单级互连网络。对于具有 N 个结点的立方体单级互连网络共有 $n = \log_2 N$ 种互联函数,即

$$\text{Cube}_i(P_{n-1} \cdots P_1 P_0) = P_{n-1} \cdots \bar{P}_i \cdots P_1 P_0$$

式中, P_i 为用二进制编号的第 i 位,且 $0 \leq i \leq n-1$ 。也就是说每一个处理单元只能与其二进制编号的某一位取反编号的处理单元相连接。

(2) PM2I 单级互连网络。这就是“加减 2^i ”单级互连网络,能实现与 j 号处理单元相连接的处理单元号为 $j \pm 2^i$ 。其互联函数为:

$$\text{PM2}_{+,i}(j) = j + 2^i \bmod N$$

$$\text{PM2}_{-,i}(j) = j - 2^i \bmod N$$

(3) 混洗交换单级互连网络。这种互联方式的互联函数为:

$$\text{Shuffle}(P_{n-1} P_{n-2} \cdots P_1 P_0) = (P_{n-2} \cdots P_1 P_0 P_{n-1})$$

可见,将处理单元的二进制编号循环左移一位便是要连接的处理单元的二进制编号。

2. 多处理机

多处理机系统是有多台处理机组成的系统。每台处理机有属于自己的控制部件,可以执行独立的程序,共享一个主存储器 and 所有的外部设备。它是多指令流多数据流计算机。在多处理机系统中,机间的互联技术决定着多处理机的性能。多处理机之间的互连,要满足高频带、低成本、连接方式的多样性以及在不规则通信情况下连接的无

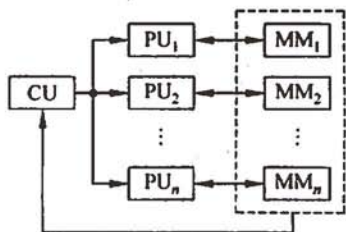


图 1-23 SIMD 计算机

冲突性。

1) 多处理机按其构成的分类

(1) 异构型(非对称型)多处理机系统: 是由多个不同类型或可完成不同功能的处理机组成, 按照作业要求的顺序, 利用时间重叠技术, 依次对它们的多个任务进行处理, 各自完成规定的功能操作。

(2) 同构型(对称型)多处理机系统: 是由多个同类型或可完成同等功能的处理机组成, 同时处理同一作业中能并行执行的多个任务。

(3) 分布式处理系统: 是把若干台具有独立功能的处理机相互连接起来, 在操作系统的控制下, 统一协调地工作, 是最少依赖集中的程序、数据或硬件的系统。

2) 多处理机系统的结构

按照机间的互连结构, 有如下 4 种多处理机结构。

(1) 总线结构: 总线结构是一种最简单的结构形式, 它把处理机与 I/O 之间的通信方式引入到处理机之间。总线式结构中有单总线结构、多总线结构、分级式总线、环式总线等多种。在此只介绍单总线结构和多总线结构。

在单总线结构中, 处理机和外部设备通过自身的接口用一套总线相连, 如图 1-24 所示。在单总线结构中, 同一时间内只允许一对处理机或设备之间进行信息的传送, 当处理机的个数较多时, 系统性能会受到严重影响, 为此, 必须采用改进方案——多总线结构。

在多总线结构中, 可以设置多套总线, 如处理机总线、存储器总线、I/O 总线等, 以增加处理机之间的通信线路, 提高处理机之间的传送效率, 如图 1-25 所示。其中, SBC 为总线控制器, I/O C 为 I/O 通道控制器。

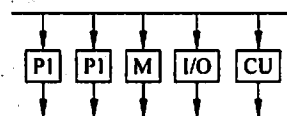


图 1-24 单总线结构

(2) 交叉开关结构: 交叉开关结构是设置一纵横开关阵列, 把横向的处理机 P 及 I/O 通道与纵向的存储器 M 连接起来的结构, 如图 1-26 所示。

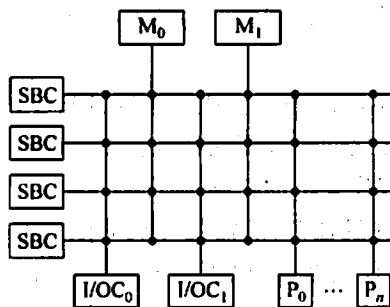


图 1-25 EPOS 多总线系统结构

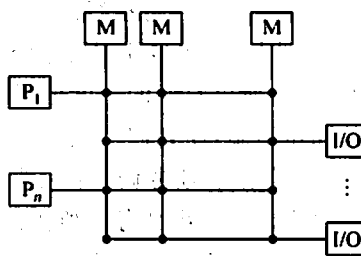


图 1-26 交叉开关式结构

(3) 多端口存储器结构：在多端口存储器结构中，将多个多端口存储器的对应端口连在一起，每一个端口负责一个处理机 P 及 I/O 通道的访问存储器的要求。如图 1-27 所示为 4 端口存储器的系统结构。

(4) 开关枢纽式结构：在开关枢纽式结构中，有多个输入端和多个输出端，在它们之间切换，使输入端有选择地与输出端相连。因为有多输入端，所以存在互连要求上的冲突。为此加入一个具有分解冲突的部件，称为仲裁单元。仲裁单元与在一个输入端和多个输出端间进行转换的开关单元一起构成一个基本的开关枢纽。任何互连网络都是由一个或多个开关枢纽组成的，如图 1-28 所示。

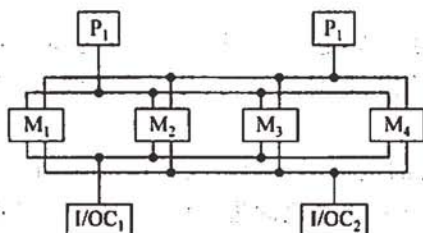


图 1-27 4 端口存储器系统结构

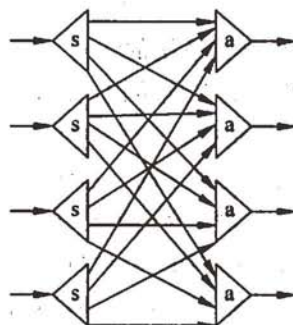


图 1-28 总线和交叉开关的枢纽形式

3) 多处理机系统的特点

(1) 结构灵活性：多处理机系统可以把要并行处理的任务、数据乃至标量都进行并行处理，所以具有较强的通用性及灵活性。

(2) 程序的并行性：在多处理机系统中，并行性表现在多个任务之间，可以利用多种途径实现并行。

(3) 并行任务派生：多处理机是多指令流、多数据流计算机，一个程序中存在多个并发的程序段，需要有专门的指令表示它们的并发关系以及控制它们的并发执行，使一个任务执行时可派生与其并行执行的另一些任务。

(4) 进程同步：在多处理机系统中，同一时刻，不同的处理机执行不同的指令。由于执行时间不等，所以它们的进度也不等。当并发程序之间有数据交往或控制依赖时，则采取特殊的同步措施，使它们所包含的指令之间保持程序要求的正确顺序。

(5) 资源分配和任务调度：因为多处理机执行并发任务，所需处理机的数目不固定，各个处理机进入或退出任务时所需的资源的情况也很复杂，所以资源分配和任务调度的好坏直接影响整个系统的效率。

3. 并行处理机

并行处理机与采用流水结构的单机系统都是单指令流多数据流计算机,但它们也有区别,并行处理机采用资源重复技术,而采用流水结构的单机系统则采用时间重叠技术。

1) 并行处理机的两种典型结构

具有分布存储器的并行处理机结构和具有共享存储器的并行处理机结构分别如图 1-29 和图 1-30 所示。

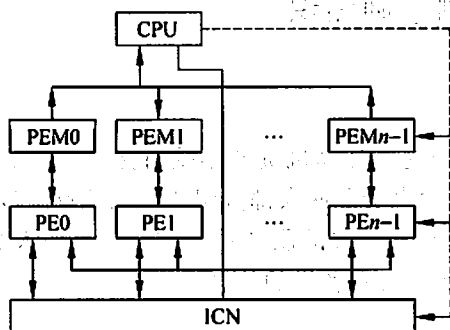


图 1-29 具有分布存储器的并行处理机结构

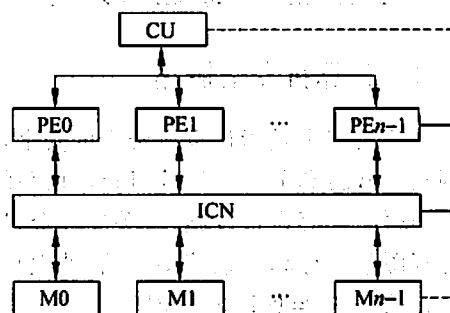


图 1-30 具有共享存储器的并行处理机结构

具有分布存储器的并行处理机结构中有两类存储器。一类存储器附属与主处理机,主处理机实现整个并行处理机的管理,在其附属的存储器内常驻操作系统。另一类是分布在各个处理单元 PE 上的存储器(即 PEM),这类存储器用来保存程序和数据。每个处理单元只与附属与自身的存储器直接相连,而各处理单元之间的通信则采用互连网络 ICN 交换数据。

说明: 图中 PE 为处理单元;CU 为控制部件;PEM 为局部存储器;M 为共享存储器;ICN 为互连网络。

在具有共享存储器的并行处理机结构中,将若干个存储器构成统一的并行处理机存储器,通过互连网络 ICN 为整个并行系统的所有处理单元共享。

上述两种结构的共同特点是在整个系统中设置多个处理单元,各个处理单元按照一定的连接方式交换信息,在统一的控制部件作用下,各自对分配来的数据并行地完成同一条指令所规定的操作。

2) 并行处理机的特点

(1) 资源重复: 并行处理机中的各个处理单元可以对向量所包含的各个分量同时进行运算;另外,每个处理单元可以承担多种处理功能。所以,增加处理单元的个数,可以提高并行处理机的运算速度。

(2) 连接模式: 并行处理机的各个处理单元之间通过互连网络交换数据,互连网络

的不同拓扑结构直接决定了并行处理机的结构。

(3) 专用性: 并行处理机直接与一定的算法相联系, 所以它具有专用性。

(4) 复合性: 并行处理机的效率体现在向量数组处理上, 所以整个系统是由 3 部分复合起来的一个多机系统; 多个处理单元组成阵列并行地处理向量; 功能极强的控制部件是一台标量处理机; 系统的管理功能则由高性能单处理机完成。

1.4 安全性、可靠性与系统性能评测基础知识

1.4.1 计算机安全概述

计算机安全是指计算机资产的安全, 是要保证这些计算机资产不受自然和人为的有害因素的威胁和危害。计算机资产由系统资源和信息资源两大部分组成。系统资源包括硬件、软件、配套设备设施、有关文件资料, 还可以包括有关的服务系统和业务工作人员。信息资源包括计算机系统中存储、处理和传输的大量各种各样的信息。

1. 信息安全的基本要素

20 世纪 80 年代后, 人们利用通信网络把孤立的单机系统连接起来, 相互通信和共享资源。随之而来的是计算机信息的安全问题: 如何保护机密信息不受黑客和工业间谍的入侵, 已成为未来信息技术中的主要问题之一。研究计算机信息安全的目的: 为了改善计算机系统和应用系统中的不可靠因素, 以保证计算机正常运行和运算结果的正确性。

信息安全的 5 个基本要素为机密性、完整性、可用性、可控性和可审计性。

(1) 机密性: 确保信息不暴露给未授权的实体或进程。

(2) 完整性: 只有得到允许的人才能修改数据, 并能够判别出数据是否已被篡改。

(3) 可用性: 得到授权的实体在需要时可访问数据。

(4) 可控性: 可以控制授权范围内的信息流向及行为方式。

(5) 可审查性: 对出现的安全问题提供调查的依据和手段。

计算机安全是一个涵盖非常广的课题, 既包括硬件、软件和技术, 还包括安全规划、安全管理和安全监督。计算机安全可包括安全管理、通信与网络安全、密码学、安全体系及模型、容错与容灾、涉及安全的应用程序及系统开发、法律、犯罪及道德规范等领域。

其中安全管理是非常重要的, 作为信息系统的管理部门应根据管理原则和该系统处理数据的保密性, 制定相应的管理制度或规范。例如, 根据工作的重要程度确定系统的安全等级; 根据确定的安全等级确定安全管理的范围, 制定相应的机房管理制度、操作规程、系统维护措施及应急措施等。

2. 计算机的安全等级

计算机系统中的 3 类安全性指技术安全性、管理安全性、政策法律安全性。但是, 一

个安全产品的购买者如何知道产品的设计是否符合规范,是否能解决计算机网络的安全问题,不同组织机构各自都制定了一套安全评估准则。一些重要的安全评估准则有:

(1) 美国国防部(DOD)和国家标准局(现更名为 NIST)的可信计算机系统评估准则;

(2) 欧洲共同体的信息技术安全评估准则(ITSEC);

(3) ISO/IEC 国际标准;

(4) 美国联邦标准。

其中,美国国防部和国家标准局的《可信计算机系统评测标准》TCSEC/TDI 将系统划分为 4 组 7 个等级,如表 1-4 所示。

表 1-4 安全性的级别

组	安全级别	定 义
1	A1	可验证安全设计; 提供 B3 级保护,同时给出系统的形式化隐秘通道分析和非形式化代码一致性验证
2	B3	安全域; 该级的 TCB 必须满足访问监控器的要求,提供系统恢复过程
	B2	结构化安全保护; 建立形式化的安全策略模型,并对系统内的所有主体和客体实施自主访问和强制访问控制
	B1	标记安全保护; 对系统的数据加以标记,并对标记的主体和客体实施强制存取控制
3	C2	受控访问控制; 实际上是安全产品的最低档次,提供受控的存取保护,存取控制以用户为单位
	C1	只提供非常初级的自主安全保护,能实现对用户和数据的分离,进行自主存取控制,数据的保护以用户组为单位
4	D	最低级别,保护措施很小,没有安全功能

3. 安全威胁

随着信息交换的激增,安全威胁所造成的危害越来越被受到重视,因此对信息保密的需求也从军事、政治和外交等领域迅速扩展到民用和商用领域。所谓安全威胁是指某个人、物、事件对某一资源的机密性、完整性、可用性或合法性所造成的危害。某种攻击就是威胁的具体实现。安全威胁分为两类:故意(如黑客渗透)和偶然(如信息发往错误的地址)。

典型的安全威胁举例如表 1-5 所示。

表 1-5 典型的安全威胁

威 胁	说 明
授权侵犯	为某一特权使用一个系统的人却将该系统用作其他未授权的目的
拒绝服务	对信息或其他资源的合法访问被无条件地拒绝,或推迟与时间密切相关的操作
窃听	信息从被监视的通信过程中泄露出去
信息泄露	信息被泄露或暴露给某个未授权的实体
截获/修改	某一通信数据项在传输过程中被改变、删除或替代
假冒	一个实体(人或系统)假装成另一个实体
否认	参与某次通信交换的一方否认曾发生过此次交换
非法使用	资源被某个未授权的人或者未授权的方式使用
人员疏忽	一个授权的人为了金钱或利益或由于粗心将信息泄露给未授权的人
完整性破坏	通过对数据进行未授权的创建、修改或破坏,使数据的一致性受到损坏
媒体清理	信息被从废弃的或打印过的媒体中获得
物理入侵	一个人侵者通过绕过物理控制而获得对系统的访问
资源耗尽	某一资源(如访问端口)被故意超负荷地使用,导致其他用户的服务被中断

4. 影响数据安全的因素

影响数据安全的因素有内部和外部两类。

(1) 内部因素。可采用多种技术对数据加密;制定数据安全规划;建立安全存储体系,包括容量、容错数据保护、数据备份等;建立事故应急计划和容灾措施;重视安全管理,制定数据安全管理制度。

(2) 外部因素。可将数据分成不同的密级,规定外部使用人员的权限。设置身份认证,设置密码、口令、指纹、声纹笔迹等多种认证。设置防火墙,为计算机建立一道屏障,防止外部入侵破坏数据。建立入侵检测、审计和追踪,对计算机进行防卫。同时,还应包括计算机的物理环境的保障、防辐射、防水、防火等外部防灾措施。

数据加密是计算机网络安全中的一个非常重要的部分。为了保证安全,不仅要对口令加密,有时还要对网上传送的文件加密。例如,为了保证电子邮件的安全,人们采用了数字签名,提供基于加密的身份认证技术。下面重点介绍数据加密和数字签名技术。

1.4.2 加密技术

1. 加密技术概述

加密技术是最常用的安全保密手段,数据加密技术的关键在于:加密/解密算法和密钥管理。加密技术包括两个元素:算法和密钥。数据加密的基本过程就是对原来为明文的文件或数据按某种加密算法进行处理,使其成为不可读的一段代码,通常称为“密文”。

“密文”只能在输入相应的密钥之后才能显示出原来内容,通过这样的途径来达到保护数据不被窃取。

数据加密和数据解密是一对逆过程。数据加密用加密算法 E 和加密密钥 K_1 ,将明文 P 变换成密文 C ,记为:

$$C=E_{K_1}(P)$$

数据解密是数据加密的逆过程。解密算法 D 和解密密钥 K_2 ,将密文 C 变换成明文 P ,记为:

$$P=D_{K_2}(C)$$

在安全保密中,可通过适当的密钥加密技术和管理机制来保证网络的信息通信安全。密钥加密技术的密码体制分为对称密钥体制和非对称密钥体制两种。相应地,对数据加密的技术分为两类,即对称加密(私人密钥加密)和非对称加密(公开密钥加密)。对称加密采用了对称密码编码技术,它的特点是文件加密和解密使用相同的密钥,即加密密钥也可以用作解密密钥。这种方法在密码学中叫做对称加密算法。对称加密算法使用起来简单快捷,密钥较短,且破译困难。对称加密以数据加密标准(Digital Encryption Standard, DES)算法为典型代表。非对称加密通常以 RSA(Rivest Shamir Adleman)算法为代表。对称加密的加密密钥和解密密钥相同,而非对称加密的加密密钥和解密密钥不同,加密密钥可以公开而解密密钥需要保密。

数据加密技术可分为对称型加密、不对称型加密和不可逆加密 3 类。

2. 对称加密技术

对称加密的体制模型如图 1-31 所示。

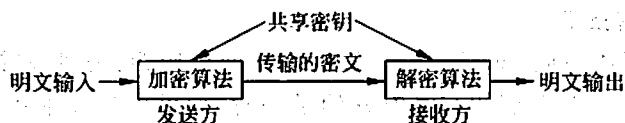


图 1-31 对称加密体制模型

下面介绍目前常用的对称加密算法。

1) 数据加密标准算法

DES 作为联邦信息处理第 46 号标准(FIPS PUB 46),1994 年美国国家标准和技术局(NIST)再次肯定 DES 以 FIPS PUB 46-2 的名义供联邦再使用 5 年。DES 主要采用替换和移位的方法加密。它用 56 位密钥对 64 位二进制数据块进行加密,每次加密可对 64 位的输入数据进行 16 轮编码,经一系列替换和移位后,输入的 64 位原始数据转换成完全不同的 64 位输出数据。DES 算法仅使用最大为 64 位的标准算术和逻辑运算,运算速度

快,密钥生产容易,适合在当前大多数计算机上用软件方法实现,同时也适合在专用芯片上实现。

2) 三重 DES(3DES,或称 TDEA)

在 DES 的基础上采用三重 DES,即用两个 56 位的密钥 K_1 、 K_2 ,发送方用 K_1 加密, K_2 解密,再使用 K_1 加密。接收方则使用 K_1 解密, K_2 加密,再使用 K_1 解密,其效果相当于将密钥长度加倍。1999 年 TDEA 合并到加密标准中,文献号为 FIPS PUB46-3。

3) RC-5(Rivest Cipher 5)

RC-5 是由 Ron Rivest(公钥算法的创始人之一)在 1994 年开发出来的。RC-5 是在 RCF2040 中定义的,RSA 数据安全公司的很多产品都使用了 RC-5。

4) 国际数据加密算法(International Data Encryption Algorithm,IDEA)

除了数据加密标准(DES),另一个对称密钥加密系统是国际数据加密算法(IDEA)。它比 DES 的加密性好,而且对计算机功能要求也没有那么高。IDEA 是瑞士的著名学者提出的,它在 1990 年正式公布并在以后得到增强。开始,该算法命名为 PES(Proposed Encryption Standard),1992 年被命名为 IDEA,目前,IDEA 被认为是相对安全的分组密码算法。

这种算法是在 DES 算法的基础上发展出来的,类似于三重 DES。发展 IDEA 也是因为感到 DES 具有密钥太短等缺点,已经过时。IDEA 的密钥为 128 位,这么长的密钥在今后若干年内应该是安全的。类似于 DES,IDEA 算法也是一种数据块加密算法。它设计了一系列加密轮次,每轮加密都使用从完整的加密密钥中生成的一个子密钥。IDEA 加密标准由 PGP(Pretty Good Privacy)系统使用。

3. 非对称加密技术

与对称加密算法不同,非对称加密算法需要两个密钥:公开密钥(publickey)和私有密钥(privatekey)。公开密钥与私有密钥是一对,如果用公开密钥对数据进行加密,只有用对应的私有密钥才能解密;如果用私有密钥对数据进行加密,那么只有用对应的公开密钥才能解密。因为加密和解密使用两个不同的密钥,所以这种算法叫做非对称加密算法。

非对称加密有两个不同的体制,如图 1-32 所示。



图 1-32 非对称加密体制模型

非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其他方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。

非对称加密算法的保密性比较好，它消除了最终用户交换密钥的需要，但加密和解密花费时间长，速度慢，不适合对文件加密而只适用对少量数据进行加密。

4. 密钥管理

密钥是有生命周期的，它包括密钥和证书的有效时间，以及已撤销密钥和证书的维护时间等。密钥既然要求保密，这就涉及到密钥的管理问题，管理不好，密钥同样可能被无意识地泄露；并不是有了密钥就高枕无忧，任何保密也只是相对的，是有时效的。密钥管理主要是指密钥对的安全管理，包括密钥产生、密钥备份、密钥恢复和密钥更新等。

1) 密钥产生

密钥对的产生是证书申请过程中重要的一步，其中产生的私钥由用户保留，公钥和其他信息则交给 CA 中心进行签名，从而产生证书。根据证书类型和应用的不同，密钥对的产生也有不同的形式和方法。对普通证书和测试证书，一般由浏览器或固定的终端应用来产生，这样产生的密钥强度较小，不适合应用于比较重要的安全网络交易。而对于比较重要的证书，如商家证书和服务器证书等，密钥对一般由专用应用程序或 CA 中心直接产生，这样产生的密钥强度大，适合重要的应用场合。

另外，根据密钥的应用不同，也可能会有不同的产生方式。比如签名密钥可能在客户端或 RA 中心产生，而加密密钥则需要在 CA 中心直接产生。

2) 密钥备份和恢复

在一个 PKI(Public Key Infrastructure, 公开密钥体系)系统中，维护密钥对的备份至关重要。如果没有这种措施，当密钥丢失后，将意味着加密数据的完全丢失，对于一些重要数据，这将是灾难性的。所以，密钥的备份和恢复也是 PKI 密钥管理中的重要一环。换句话说，即使密钥丢失，使用 PKI 的企业和组织必须仍能够得到确认，受密钥加密保护的重要信息也必须能够恢复。当然，不能让一个独立的个人完全控制最重要的主密钥，否则将引起严重后果。

企业级的 PKI 产品至少应该支持用于加密的安全密钥的存储、备份和恢复。密钥一般用口令进行保护，而口令丢失则是管理员最常见的安全疏漏之一。所以，PKI 产品应该能够备份密钥，即使口令丢失，它也能够让用户在一定条件下恢复该密钥，并设置新的口令。

3) 密钥更新

如果用户可以一次又一次地使用同样密钥与别人交换信息，那么密钥也同其他任何

密码一样存在着一定的安全性。虽然说用户的私钥是不对外公开的,但是也很难保证私钥长期的保密性,很难保证长期以来不被泄露。如果某人偶然地知道了用户的密钥,那么用户曾经和另一个人交换的每一条消息都不再是保密的了。另外,使用一个特定密钥加密的信息越多,提供给窃听者的材料也就越多,从某种意义上讲也就越不安全了。

对每一个由 CA 颁发的证书都会有有效期。密钥对生命周期的长短由签发证书的 CA 中心来确定,各 CA 系统的证书有效期限有所不同,一般大约为 2~3 年。当用户的私钥被泄露或证书的有效期快到时,用户应该更新私钥。这时,用户可以废除证书,产生新的密钥对,申请新的证书。

4) 多密钥的管理

假设在某机构中有 100 个人,如果他们任意两人之间可以进行秘密对话,那么总共需要多少密钥呢,每个人需要知道多少密钥呢。也许很容易得出答案,如果任何两个人之间要有不同的密钥,则总共需要 4950 个密钥,而且每个人应记住 99 个密钥。如果机构的人数是 1000 人、10 000 人或更多,这种办法就显然过于愚蠢了,管理密钥将是一件非常困难的事情。为此需要研究并开发用于创建和分发密钥的加密安全的方法。

Kerberos 提供了一种较好的解决方案,它是由 MIT 发明的,使保密密钥的管理和分发变得十分容易,但这种方法本身还存在一定的缺点。为能在因特网上提供一个实用的解决方案,Kerberos 建立了一个安全的、可信任的密钥分发中心(Key Distribution Center, KDC),每个用户只要知道一个和 KDC 进行会话的密钥就可以了,而不需要知道成百上千个不同的密钥。

1.4.3 认证技术

1. 认证技术概述

认证技术主要解决网络通信过程中通信双方的身份认可。认证的过程涉及到加密和密钥交换。通常,加密可使用对称加密、不对称加密及两种加密方法的混合方法。认证方一般有账户名/口令认证、使用摘要算法认证和基于 PKI(Public Key Infrastructure, 公开密钥体系)的认证。

一个有效的 PKI 系统必须是安全的和透明的,用户在获得加密和数字签名服务时,不需要详细地了解 PKI 的内部运作机制。在一个典型、完整和有效的 PKI 系统中,除证书的创建和发布,特别是证书的撤销以外,一个可用的 PKI 产品还必须提供相应的密钥管理服务,包括密钥的备份、恢复和更新等。没有一个好的密钥管理系统,将极大影响一个 PKI 系统的规模、可伸缩性和在协同网络中的运行成本。在一个企业中,PKI 系统必须有能力为一个用户管理多对密钥和证书;能够提供安全策略编辑和管理工具,如密钥周期和密钥用途等。

PKI 是一种遵循既定标准的密钥管理平台,它能够为所有网络应用提供加密和数字签名等密码服务及所必需的密钥和证书管理体系。简单地说,PKI 就是利用公钥理论和技术建立的提供安全服务的基础设施。PKI 技术是信息安全技术的核心,也是电子商务的关键和基础技术。PKI 的基础技术包括加密、数字签名、数据完整性机制、数字信封、双重数字签名等。完整的 PKI 系统必须具有权威认证机构(CA)、数字证书库、密钥备份及恢复系统、证书作废系统、应用接口(API)等基本构成部分。

(1) 认证机构(CA): 即数字证书的申请及签发机关,CA 必须具备权威性的特征。

(2) 数字证书库: 用于存储已签发的数字证书及公钥,用户可由此获得所需的其他用户的证书及公钥。

(3) 密钥备份及恢复系统: 如果用户丢失了用于解密数据的密钥,则数据将无法被解密,这将造成合法数据丢失。为避免这种情况,PKI 提供备份与恢复密钥的机制。但须注意,密钥的备份与恢复必须由可信的机构来完成。并且,密钥备份与恢复只能针对解密密钥,签名私钥为确保其惟一性而不能作备份。

(4) 证书作废系统: 证书作废处理系统是 PKI 的一个必备的组件。与日常生活中的各种身份证件一样,证书有效期以内也可能需要作废,原因可能是密钥介质丢失或用户身份变更等。为实现这一点,PKI 必须提供作废证书的一系列机制。

(5) 应用接口(API): PKI 的价值在于使用户能够方便地使用加密、数字签名等安全服务,因此一个完整的 PKI 必须提供良好的应用接口系统,使得各种各样的应用能够以安全、一致、可信的方式与 PKI 交互,确保安全网络环境的完整性和易用性。

PKI 采用证书进行公钥管理,通过第三方的可信任机构(认证中心,即 CA),把用户的公钥和用户的其他标识信息捆绑在一起,其中包括用户名和电子邮件地址等信息,以在 Internet 网上验证用户的身份。PKI 把公钥密码和对称密码结合起来,在 Internet 网上实现密钥的自动管理,保证网上数据的安全传输。

因此,从大的方面来说,所有提供公钥加密和数字签名服务的系统,都可归结为 PKI 系统的一部分。PKI 的主要目的是通过自动管理密钥和证书,为用户建立起一个安全的网络运行环境,使用户可以在多种应用环境下方便地使用加密和数字签名技术,从而保证网上数据的机密性、完整性、有效性。数据的机密性是指数据在传输过程中,不能被非授权者偷看;数据的完整性是指数据在传输过程中不能被非法篡改;数据的有效性是指数据不能被否认。

PKI 发展的一个重要方面就是标准化问题,它也是建立互操作性的基础。目前,PKI 标准化主要有两个方面: 一是 RSA 公司的公钥加密标准 PKCS (Public Key Cryptography Standards),它定义了许多基本 PKI 部件,包括数字签名和证书请求格式等;二是由 Internet 工程任务组 IETF(Internet Engineering Task Force)和 PKI 工作组

PKIX(Public Key Infrastructure Working Group)所定义的一组具有互操作性的公钥基础设施协议。在今后很长的一段时间内,PKCS 和 PKIX 将会并存,大部分的 PKI 产品为保持兼容性,也会对这两种标准进行支持。

2. HASH 函数与信息摘要(message digest)

HASH(哈希)函数提供了这样一种计算过程:输入一个长度不固定的字符串,返回一串固定长度的字符串,又称 HASH 值。单向 HASH 函数用于产生信息摘要。HASH 函数主要可以解决以下两个问题:在某一特定的时间内,无法查找经 HASH 操作后生成特定 HASH 值的原文;也无法查找两个经 HASH 操作后生成相同 HASH 值的不同报文。这样,在数字签名中就可以解决验证签名和用户身份验证、不可抵赖性的问题。

信息摘要简要地描述了一份较长的信息或文件,它可以被视为一份长文件的“数字指纹”。信息摘要用于创建数字签名,对于特定的文件而言,信息摘要是惟一的。信息摘要可以被公开,它不会透露相应文件的任何内容。MD2,MD4 和 MD5(MD 表示信息摘要)是由 Ron Rivest 设计的专门用于加密处理的,并被广泛使用的 HASH 函数。它们产生一种 128 位信息摘要,除彻底地搜寻外,没有更快的方法对其加以攻击,而其搜索时间一般需要 1025 年之久。

3. 数字签名

数字签名主要经过以下几个过程:

- ① 信息发送者使用一单向散列函数(hash 函数)对信息生成信息摘要;
- ② 信息发送者使用自己的私钥签名信息摘要;
- ③ 信息发送者把信息本身和已签名的信息摘要一起发送出去;
- ④ 信息接收者通过使用与信息发送者使用的同一个单向散列函数(HASH 函数)对接收的信息本身生成新的信息摘要,再使用信息发送者的公钥对信息摘要进行验证,以确认信息发送者的身份和信息是否被修改过。

数字加密主要经过以下几个过程:

- ① 当信息发送者需要发送信息时,首先生成一个对称密钥,用该对称密钥加密要发送的报文;
- ② 信息发送者用信息接收者的公钥加密上述对称密钥;
- ③ 信息发送者将第一步和第二步的结果结合在一起传给信息接收者,称为数字信封;
- ④ 信息接收者使用自己的私钥解密被加密的对称密钥,再用此对称密钥解密被发送方加密的密文,得到真正的原文。

数字签名和数字加密的过程虽然都使用公开密钥体系,但实现的过程正好相反,使用的密钥对也不同。数字签名使用发送方的密钥对,发送方用自己的私有密钥进行加密,接

收方用发送方的公开密钥进行解密。这是一个一对多的关系,任何拥有发送方公开密钥的人都可以验证数字签名的正确性。数字加密则使用接收方的密钥对,这是多对一的关系,任何知道接收方公开密钥的人都可以向接收方发送加密信息,只有惟一拥有接收方私有密钥的人才能对信息解密。另外,数字签名只采用非对称密钥加密算法,它能保证发送信息的完整性、身份认证和不可否认性;而数字加密采用对称密钥加密算法和非对称密钥加密算法相结合的方法,它能保证发送信息的保密性。

4. SSL 安全协议

SSL 安全协议最初是由 Netscape Communication 公司设计开发的,又叫“安全套接层(Secure Sockets Layer)协议”,主要用于提高应用程序之间数据的安全系数。SSL 协议的整个概念可以被总结为:一个保证任何安装了安全套接层的客户和服务端间事务安全的协议,它涉及所有 TCP/IP 应用程序。SSL 安全协议主要提供三方面的服务:

(1) 用户和服务器的合法性认证。认证用户和服务器的合法性,使得它们能够确信数据将被发送到正确的客户机和服务器上。客户机和服务器都有各自的识别号,这些识别号由公开密钥进行编号。为了验证用户是否合法,安全套接层协议要求在握手交换数据时进行数字认证,以此来确保用户的合法性。

(2) 加密数据以隐藏被传送的数据。安全套接层协议所采用的加密技术既有对称密钥技术,也有公开密钥技术。在客户机与服务器进行数据交换之前,交换 SSL 初始握手信息,在 SSL 握手信息中采用了各种加密技术对其加密,以保证其机密性和数据的完整性,并且用数字证书进行鉴别,这样就可以防止非法用户进行破译。

(3) 保护数据的完整性。安全套接层协议采用 hash 函数和机密共享的方法来提供信息的完整性服务,建立客户机与服务器之间的安全通道,使所有经过安全套接层协议处理的业务在传输过程中能全部、完整、准确无误地到达目的地。

安全套接层协议是一个保证计算机通信安全的协议,对通信对话过程进行安全保护,其实现过程主要经过如下几个阶段:

- ① 接通阶段:客户机通过网络向服务器打招呼,服务器回应;
- ② 密码交换阶段:客户机与服务器之间交换双方认可的密码,一般选用 RSA 密码算法,也有的选用 Diffie-Hellman 和 Fortezza-KEA 密码算法;
- ③ 会谈密码阶段:客户机与服务器间产生彼此交谈的会谈密码;
- ④ 检验阶段:客户机检验服务器取得的密码;
- ⑤ 客户认证阶段:服务器验证客户机的可信度;
- ⑥ 结束阶段:客户机与服务器之间相互交换结束的信息。

当上述动作完成之后,两者间的资料传送就会加密;另外一方收到资料后,再将编码资料还原。即使窃窃者在网络上取得编码后的资料,如果没有原先编制的密码算法,也不

能获得可读的有用资料。

发送时信息用对称密钥加密,对称密钥用非对称算法加密,再把两个包绑在一起传过去。接收的过程与发送正好相反,先打开有对称密钥的加密包,再用对称密钥解密。

在电子商务交易过程中,由于有银行参与,按照 SSL 协议,客户的购买信息首先发往商家,商家再将信息转发银行。银行验证客户信息的合法性后,通知商家付款成功,商家再通知客户购买成功,并将商品寄送客户。

5. 数字时间戳技术

数字时间戳技术就是数字签名技术的一种变种。在电子商务交易文件中,时间是十分重要的信息。在书面合同中,文件签署的日期和签名一样均是十分重要的防止文件被伪造和篡改的关键性内容。数字时间戳服务(digital time stamp service, DTS)是电子商务安全服务项目之一,能提供电子文件的日期和时间信息的安全保护。

时间戳(time-stamp)是一个经加密后形成的凭证文档,它包括 3 个部分:

- (1) 需加时间戳的文件的摘要(digest);
- (2) DTS 收到文件的日期和时间;
- (3) DTS 的数字签名。

一般来说,时间戳产生的过程为:用户首先将需要加时间戳的文件用 hash 编码加密形成摘要,然后将该摘要发送到 DTS。DTS 在加入了收到文件摘要的日期和时间信息后再对该文件加密(数字签名),然后送回用户。

书面签署文件的时间是由签署人自己写上的,而数字时间戳则不然,它是由认证单位 DTS 来加的,以 DTS 收到文件的时间为依据。

1.4.4 计算机病毒的防治

计算机病毒是近十多年来发展起来的一种新的计算机犯罪形式。计算机病毒造成的危害越来越严重,已经引起普遍关注。计算机病毒的检测、消除和预防,已经成为计算机系统日常安全维护工作中非常重要的一个方面。在我国,IBM PC 系列微机(包括兼容机)已拥有数量极大的用户。目前大量微机是“带病毒工作”的,大量软件和数据是通过带有病毒的介质传送的;具有防病毒能力的微机系统仍然被计算机病毒攻击,病毒防护工作赶不上病毒的发展。计算机网络的迅速发展,同时也为大范围地快速地传染计算机病毒提供了更为便利的条件。这些情况应当受到足够的重视。

本节的讨论主要针对用户最多、病毒发生最普遍的 IBM PC 系列微机(含兼容机)。

1. 计算机病毒的定义和特点

计算机病毒(computer virus)是一种程序,它具有这样的特性:它可以修改别的程序,使得被修改的程序也具有这种特性。

病毒程序不同于通常程序,它具有以下特点:

(1) 寄生性。病毒程序的存在不是独立的,它总是悄悄地附着在磁盘系统区或文件中。寄生于文件中的病毒是文件型病毒。其中病毒程序附在原来文件之前或之后的,称为文件外壳型病毒,如以色列病毒(黑色星期五)等。另一种文件型病毒为嵌入型,其病毒程序嵌入到原来文件之中,在微机病毒中尚未见到。病毒程序侵入磁盘系统区的称为系统型病毒,其中较常见的占据引导区的病毒,称为引导区病毒,如大麻病毒、2708 病毒等。此外,还有一些既寄生于文件中又侵占系统区的病毒,如“幽灵”病毒、Flip 病毒等,属于混合型病毒。

(2) 隐蔽性。这种病毒程序在一定条件下能隐蔽地进入系统。当使用带有系统病毒的磁盘来引导系统时,病毒程序先进入内存并放在常驻区,然后才引导系统,这时系统即带有该病毒。当运行带有病毒的程序文件(com 文件或 exe 文件,有时包括覆盖文件)时,先执行病毒程序,然后才执行该文件的原来程序。有的病毒将自身程序常驻内存,使系统成为病毒环境;有的病毒则不常驻内存,只在执行当时进行传染或破坏,执行完毕之后病毒不再留在系统中。

(3) 非法性。病毒程序执行的是非授权(非法)操作。当用户引导系统时,正常的操作只是引导系统,病毒乘机而入,并不在人们预定目标之内。

(4) 传染性。病毒程序在其运行过程中进行自我复制,寻找适宜的介质或文件作为新的寄生对象,将病毒程序寄生到该介质或文件中,通常称为传染。病毒一旦进入系统,就会很快地传染到每个可能受到传染的介质或文件中,并通过这些介质或文件中,或者通过网络传染给其他计算机系统。

(5) 破坏性。病毒程序无论是静态的或是动态的,都占用一定的系统资源。同时,几乎所有的病毒程序都有自己的“表现”,轻则出现异常显示或声响,破坏操作环境,重则破坏一部分乃至全部系统资源,诸如破坏文件,使数据丢失,甚至使整个磁盘“丢失”,造成重大损失。

在《中华人民共和国计算机信息系统安全保护条例》中计算机病毒被明确定义为:“编制或者在计算机程序中插入的破坏计算机功能或者破坏数据,影响计算机使用并且能够自我复制的一组计算机指令或者程序代码”。

2. 计算机病毒的类型

计算机病毒可以有不同的分类方法。按照病毒程序的寄生方式和它对于系统的侵入方式,通常将微机病毒分成下述几类。

(1) 系统引导型病毒:病毒寄生于磁盘介质上用来引导系统的引导区(BOOT 区或硬盘主引导区),借助系统引导过程进入系统,常称为 BOOT 型病毒。

引导型病毒按照侵占磁盘主引导区的方式,又分为迁移型和替代型两种。将原来

BOOT 区内容(引导程序及磁盘参数表)和主引导区内容(主引导程序及硬盘分区表)移到磁盘中的其他扇区中,病毒程序超过 512 字节的部分也一起移放,这种病毒称为迁移型病毒,如小球病毒、大麻病毒等。如果病毒取消被侵入扇区的原有内容,而将使用磁盘所必须用的参数和程序段落纳入到病毒程序中,如 Ctrl+Break 病毒、Torch 病毒,称为替代型病毒。2708 病毒对软盘的感染采用前一种方式,对于硬盘的感染则采用后一种方式。

引导型病毒进入系统,一定要通过启动过程。在无病毒环境下使用的软盘或硬盘,即使它已感染引导区病毒,也不会进入系统并进行传染。但是,只要用感染引导区病毒的磁盘引导系统,就会使病毒程序进入内存,形成病毒环境。有的软盘 BOOT 区感染病毒后,虽然由于缺少系统文件不能用于引导系统,但是,如果启动时将这种盘放在 A 驱动器中,往往也会使系统感染病毒。

在病毒环境下使用的软盘,凡是未贴写保护签的,都可能被病毒感染。硬盘则由于无法贴写保护签,最容易被感染,并且成为继续传染其他软盘的病毒源,病毒又可以通过软盘再传染到其他计算机系统。因此,病毒往往能够很快地扩散,蔓延至更大范围,在短时间内造成大面积损害。

(2) 文件外壳型病毒:此种病毒寄生于程序文件的前边或后边,当装入程序文件并执行该程序时,病毒程序进入系统并首先被执行。

文件型病毒最显著的特点是增加文件的长度,有些病毒还改变原来文件的生成日期或时间。这些特点可以使人们通过检查文件长度、日期或时间有无变化来检测是否感染病毒。

当执行已感染病毒的程序文件(com 文件、exe 文件,有时还包括覆盖文件 ovr 等)时,病毒程序和原来程序一起装入内存,随后执行病毒程序,然后才执行该文件原来的程序。因此,从运行过程中看,执行带病毒的程序文件,相当于在执行正常操作之前插入一段病毒程序的执行过程(注:所谓“执行文件”是一种简称,确切地说,应当是“装入文件,并执行存放于该文件中的程序”)。

文件型病毒可分为瞬时作用和驻留内存两种。瞬时作用的病毒程序不驻留内存,它仅仅在执行带病毒的程序时执行一次,病毒的传染和破坏都仅仅在此期间发生。如果在此期间系统和文件未被破坏,就可以继续工作,执行原来的程序。而对于后一种病毒,一旦执行病毒程序,它就首先将自身存放于内存驻留区,同时修改系统参数和设置,包括减少用户可用内存的大小,修改中断服务程序的入口地址等,使系统从此处于病毒环境。这种环境下执行的程序文件都可能被传染上病毒,成为带病毒的程序文件。而在这种环境下往往检查不到病毒的存在,也无法实现对病毒文件的复原(即消除病毒)。

文件型病毒对文件的感染可分为主动攻击和执行时感染两种方式。主动攻击方式是

当执行病毒程序时,它检查磁盘上其他文件,如发现所检查的文件没有被感染,就立即将它感染上该病毒。瞬时作用的维也纳病毒(648 病毒)和驻留内存的 1575/1591 病毒都属于这一方式。执行时感染方式是指,在病毒环境中每当执行一个程序文件,如果所在磁盘没有写保护,这个程序文件就会感染病毒。无论哪一种方式,首先受到攻击的通常是 command.com 文件。因此,经常检查该文件的长度和日期有无变化,是检测病毒的一种方法。而有些病毒程序为了隐蔽自身,又常常规定不传染 command.com 文件。

受攻击的程序文件类型,对于不同的病毒是不相同的,有些病毒只感染 com 文件,有些病毒对 com 文件和 exe 文件都感染,还有一些文件型病毒感染其他文件。

(3) 混合型病毒:混合型病毒在寄生方式、进入系统方式和传染方式上,兼有系统引导型病毒和文件外壳型病毒两者的特点。通常,病毒传染并寄生于硬盘主引导区和程序文件中。在硬盘系统区有病毒的情况下由硬盘引导系统,使得产生病毒环境,在这一环境下执行的程序文件都会传染病毒。当病毒程序文件通过软盘带到其他计算机系统上使用时,就会使病毒进入系统,并首先传染给该系统的硬盘主引导区。病毒的这种传染过程,比单一的引导型病毒和文件型病毒都要快得多,Flip 病毒、One Half 病毒(幽灵)等都属于这一类。

(4) 目录型病毒:这一类型病毒通过装入与病毒相关的文件进入系统,而不改变相关文件,它所改变的只是相关文件的目录项。

(5) 宏病毒:Windows Word 宏病毒利用 Word 提供的宏功能,将病毒程序插入到带有宏的 doc 文件或 dot 文件中。这类病毒种类很多,传播速度很快,往往对系统或文件造成破坏。目前发现的 Word 宏病毒经常在 Word 的文档和模板范围内运行和传播。在提供宏功能的其他软件中也有宏病毒,如 Excel 宏病毒。有些学者将宏病毒归类于数据病毒。

3. 计算机病毒的繁衍

计算机病毒的种类正在以越来越快的速度增加,同时,病毒程序的伪装也越来越隐蔽,这就对病毒防护提出了更高的要求。特别值得注意的有以下几种情况。

(1) 变种:对某个病毒程序做很少的改变,例如修改作为辨认标记的几个特征代码,或者改变病毒程序在介质或文件中寄生的位置,或者改变病毒激活时的“表现”形式,用这种方法产生的病毒,通常不认为是一个新的种类,而称为是原来病毒的变体或变种。有些病毒的变体可以多达十多种或更多一些。如果原来病毒已经能够检测出来,那么,它的变体用已有的工具(软件或硬件)则有可能检测不出来。

(2) 病毒程序加密:很多病毒采用源代码加密的方法。病毒程序中除了很短小的一段解密程序之外,其他绝大部分代码都经过加密处理,增加了辨认后分析病毒程序的难度。但是,使用这种方法的病毒程序,其固定的解密子程序是可以辨认的。

(3) 多形性病毒：这种病毒进行传染时，并不是病毒程序自身的简单复制，而是采用一些复杂的方法，生成与原来病毒程序不完全一样甚至完全不同的病毒程序，附在新的介质或文件上，而它们仍然是同一种病毒，“多形性”的名称即由这一特点而来。这种病毒也是进行加密处理的，但是，它的解密部分在形式上也是各个不同的。多形性病毒首次出现于 1990 年(Flip 病毒)，1994 年肆虐一时的 One Half 病毒(幽灵 3544)、Casper 病毒(1200)等都是多形性病毒。有的资料介绍，有的病毒的形态可以有几十种至上亿种，每两个不同形态之间，没有三个连续代码是相重复的。

(4) 伪装：很多病毒都用各种方法伪装自己，欺骗用户。有的文件型病毒特意不传染 command.com 文件。系统感染某些病毒后，用 DIR 命令列文件目录时，已传染病毒的文件仍然显示原来的文件长度和日期、属性等，而不显示其真实的文件长度等信息。有的引导型病毒，只改变引导扇区的几个字节内容，很容易混同于正常内容。还有些病毒程序采用了反跟踪技术，增加了利用 DEBUG 等软件剖析病毒程序的难度。

4. 网络环境下的病毒问题

计算机网络使得网上计算机之间数据共享和数据传输成为现实。同时，网络系统本身的弱点和网络软件的缺陷，也为计算机病毒的快速传播和破坏提供了方便条件，严重危及网络系统的正常工作以及数据的安全性和保密性。事实上，首次引起人们高度重视计算机病毒问题的，正是发生于 1988 年 11 月 2 日的 Internet 网络事件，当时在网上的约 6000 台计算机受到感染。有人评估，计算机系统的直接损失高达 9600 万美元。

严格说来，网络病毒特指那些以计算机网络协议和体系结构作为传播途径并对网络系统进行破坏的病毒，例如攻击 Internet(因特网)的 Worm(蠕虫)病毒，攻击 DECnet 网络的 WANK 病毒，攻击 NetWare 网络的 GRI 病毒等。但是，一般认为，凡是能够通过网络进行传播并进行破坏的，也都可以当成网络病毒。后一种划定范围比较宽，包括绝大部分单机病毒。

网络系统中的数据通信功能，往往成为病毒传播的途径。例如，电子邮件可以传送任何类型的文件，当然也包括带有病毒的文件。通过这种方式，单机上所有由文件传送的病毒，都可以在网络上传播。这些病毒在文件传送过程中处于静态，一旦接收文件的工作站打开或运行带有病毒的文件，病毒就开始起作用了。因此，网上的每个工作站，也应当和单机一样，特别注意病毒的防范。

目前，网络系统大多采用服务器/工作站方式，在这种情况下，对于网络病毒，应当从服务器和工作站两方面都进行防治。例如，局域网 NetWare 和 Windows NT 网络系统，其防病毒体系是由网络服务器上的防病毒模块和各个用户终端上的单机防病毒模块这两部分组成的。

计算机网络的飞速发展，使得网络系统在各个应用领域中的作用越来越重要，很多工

作对网络的依赖程度越来越高,这就要求人们更加重视网络系统的病毒问题。

5. 计算机病毒防治

有些计算机病毒是非常危险的,当它们传染时会引起无法预料的和灾难性的破坏。例如,这类病毒删除程序,破坏数据,清除系统内存区和操作系统中重要的信息。因此,不仅要预防计算机病毒,而且,当发现病毒时应及时清除。

1) 人工预防

人工预防也称免疫性预防法。因为任何一种病毒都有标志,修改程序,使病毒标志固定在程序的相应位置上,达到免疫的功能。

2) 软件预防

目前主要使用计算机病毒的疫苗程序,这些程序能监督系统运行,防止某些病毒入侵。例如,计算机病毒的疫苗程序发现磁盘或内存有变化时,立即通知用户采取相关措施。

3) 管理预防

管理预防主要包括建立相应的法律制度、教育管理和建立计算机系统管理制度。

(1) 建立相应的法律制度。规定制造计算机病毒是违法的行为,对罪犯以法律制裁。

(2) 教育管理。要教育机房工作人员严格遵守制度,不私留病毒样品,防止有意或无意扩散病毒。

(3) 计算机系统管理制度。采用严密的计算机系统管理制度,可以有效地防止病毒入侵。这些制度主要有:

- 定期检查硬盘及所用到的软盘,清除内部病毒。
- 采用软件防护措施,如磁盘免疫。
- 一律用硬盘引导操作系统。如果必须用软盘引导,应保证软盘无毒。
- 禁止外来人员使用机器。
- 禁止用计算机做专业以外的其他用途(如软件开发、玩游戏等)。
- 所有重要软件都要留副本。
- 重要的数据经常备份。
- 新引进的软件必须经过确认,不带病毒方可使用。

6. 解决网络安全问题的一些技术和方法

解决网络安全问题的技术包括以下一些。

1) 划分网段、局域网交换技术和 VLAN 实现

划分网段、局域网交换技术和 VLAN 实现主要解决局域网络的安全问题。由于局域网是广播型网络,因此,若在广播域中进行监听,就可以对信息包进行分析,那么本广播域的信息传递都会暴露无遗。

划分网段,其本质就是限制广播域,将非法用户与网络资源相互隔离,从而达到限制用户非法访问的目的。其方式可分为物理分段和逻辑分段两种。物理分段通常是将网络从物理层和数据链路层上分开网段,各网段相互间无法进行直接通信;逻辑分段是指将整个系统在网络层上进行分段。

2) 加密技术、数字签名和认证、VPN 技术

加密型网络安全技术的基本思想是不依赖于网络中数据路径的安全性来实现网络系统的安全,而是通过对网络数据的加密来保障网络的安全可靠性。因而这一类安全保障技术的基石是适用的数据加密技术及其在分布式系统中的应用。

3) 防火墙技术(参见第 5 章)

4) 入侵检测技术

利用防火墙技术,通常能够在内外网之间提供安全的网络保护,降低网络的安全风险。但是,仅仅利用防火墙,网络安全还远远不够,因为入侵者可寻找防火墙背后可能敞开的后门,或入侵者可能就在防火墙内等情况发生。

入侵检测系统是新型的网络安全技术,目的是提供实时的入侵检测及采取相应的防护手段。实时入侵检测的能力重要点在于它能够对付来自内部的攻击,能够阻止 hacker 的入侵。入侵检测系统常分为基于主机和网络两类。

5) 网络安全扫描技术

网络安全扫描技术可对网络系统的安全作预先的检测,查找安全漏洞,提供解决方案等。

1.4.5 计算机可靠性

1. 计算机可靠性概述

计算机系统的硬件故障通常是由元器件的失效引起的。对元器件进行寿命试验并根据实际资料统计得知,元器件的可靠性可分成 3 个阶段。开始阶段,器件工作处于不稳定期,失效率较高;第二阶段,器件进入正常工作期,失效率最低,基本保持常数;第三阶段,元器件开始老化,失效率又重新提高;这就是所谓的“浴盆曲线”。因此,应保证在计算机中使用的元器件处于第二阶段。在第一阶段,对元器件应进行老化筛选,而到了第三个阶段,则淘汰该计算机。

计算机系统的可靠性是指从它开始运行($t=0$)到某时刻 t 这段时间内能正常运行的概率,用 $R(t)$ 表示。所谓失效率是指单位时间内失效的元件数与元件总数的比例。以 λ 表示,当 λ 为常数时,可靠性与失效率的关系为:

$$R(t) = e^{-\lambda t}$$

典型的失效率与时间的关系曲线如图 1-33 所示。

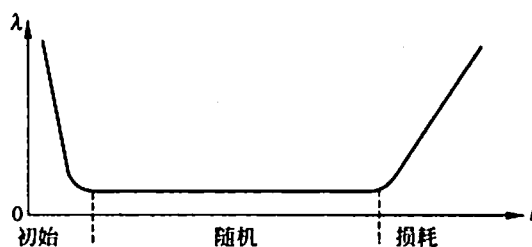


图 1-33 失效率特性

两次故障之间系统能正常工作的时间的平均值称为平均无故障时间(MTBF):

$$MTBF = 1 / \lambda$$

通常用平均修复时间(MTRF)来表示计算机的可维修性,即计算机的维修效率,指从故障发生到机器修复平均所需要的时间。计算机的可用性是指计算机的使用效率,它以系统在执行任务的任意时刻能正常工作的概率 A 来表示:

$$A = \frac{MTBF}{MTBF + MTRF}$$

计算机的 RAS 技术,就是指用可靠性(R)、可用性(A)和可维修性(S)这 3 个指标衡量一个计算机系统。但在实际应用中,引起计算机故障的原因除了元器件以外还与组装工艺、逻辑设计等因素有关。因此不同厂家生产的兼容机,即使采用相同的元器件,其可靠性及 MTBF 也可能会相差很大。

2. 计算机可靠性模型

计算机系统是一个复杂的系统,而且影响其可靠性的因素也非常复杂,很难直接对其进行可靠性分析;但通过建立适当的数学模型,把大系统分割成若干子系统,可以简化其分析过程。常见的系统可靠性数学模型有以下 3 种。

(1) 串联系统:假设一个系统由 N 个子系统组成,当且仅当所有的子系统都能正常工作时,系统才能正常工作,这种系统称为串联系统,如图 1-34 所示。设系统各个子系统的可靠性分别用 R_1, R_2, \dots, R_N 来表示,则系统的可靠性 R 可由下式求得:

$$R = R_1 R_2 \dots R_N$$

如果系统的各个子系统的失效率分别用 $\lambda_1, \lambda_2, \dots, \lambda_N$ 来表示,则系统的失效率 λ 可由下式求得:

$$\lambda = \lambda_1 + \lambda_2 + \dots + \lambda_N$$

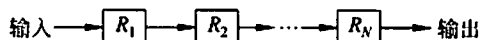


图 1-34 串联系统的可靠性模型

【例 1.1】 设计计算机系统由 CPU、存储器、I/O 3 部分组成,其可靠性分别为 0.95、0.90 和 0.85,求计算机系统的可靠性。

解: $R = R_1 \cdot R_2 \cdot R_3 = 0.95 \times 0.90 \times 0.85 = 0.73$

计算机系统的可靠性为 0.73。

(2) 并联系统: 假如一个系统由 N 个子系统组成,只要有一个子系统正常工作,系统就能正常工作,这样的系统称为并联系统,如图 1-35 所示。设每个子系统的可靠性分别以 R_1, R_2, \dots, R_N 表示,整个系统的可靠性可由下式求得:

$$R = 1 - (1 - R_1)(1 - R_2) \cdots (1 - R_N)$$

假如所有的子系统的失效率均为 λ ,则系统的失效率 μ 为:

$$\mu = \frac{1}{\frac{1}{\lambda} \sum_{j=1}^N \frac{1}{j}}$$

在并联系统中只有一个子系统是真正需要的,其余 $N-1$ 个子系统称为冗余子系统。随着冗余子系统数量的增加,系统的平均无故障时间也增加了。

【例 1.2】 设一个系统由 3 个相同子系统构成,其可靠性为 0.9,平均无故障时间为 10 000 小时,求系统的可靠性和平均无故障时间。

解: $R_1 = R_2 = R_3 = 0.9 \quad \lambda_1 = \lambda_2 = \lambda_3 = 1/10\,000 = 1 \times 10^{-4} (\text{小时})$

系统可靠性 $R = 1 - (1 - R_1)^3 = 0.999$

系统平均无故障时间 MTBF 为:

$$\text{MTBF} = \frac{1}{\mu} = \frac{1}{\lambda} \sum_{j=1}^3 \frac{1}{j} = \frac{1}{\lambda} \times \left(1 + \frac{1}{2} + \frac{1}{3}\right) = 18\,333 (\text{小时})$$

(3) N 模冗余系统: N 模冗余系统由 N 个 ($N=2n+1$) 相同的子系统和一个表决器组成,表决器把 N 个子系统中占多数相同结果的输出作为系统的输出,如图 1-36 所示。

在 N 个子系统中,只要有 $n+1$ 个或 $n+1$ 个以上子系统能正常工作,系统就能正常工作,输出正确的结果。假设表决器是完全可靠的,每个子系统的可靠性为 R_0 ,则 N 模冗余系统的可靠性为:

$$R = \sum_{i=n+1}^N \binom{N}{i} \times R_0^i (1 - R_0)^{N-i}$$

其中, $\binom{j}{N}$ 表示从 N 个元素中取 j 个元素的组合数。

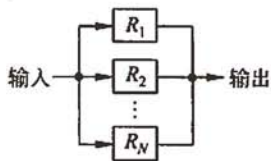


图 1-35 并联系统的可靠性模型

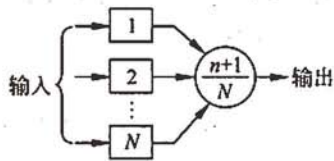


图 1-36 N 模冗余系统

提高计算机的可靠性一般采取两项措施：

- 提高元器件质量,改进加工工艺与工艺结构,完善电路设计;
- 发展容错技术,使得在计算机硬件有故障的情况下,计算机仍能继续运行,得出正确的结果。

1.4.6 计算机系统的性能评价

无论生产计算机的厂商还是使用计算机的用户都需要有某种方法来衡量计算机的性能,作为设计、生产、购买和使用的依据。但是,由于计算机系统是一个极复杂的系统,其体系结构、组成和实现都有若干种策略,而且其应用领域也千差万别,所以很难找到统一的规则或标准去评测所有的计算机。

1. 性能评测的常用方法

(1) 时钟频率: 计算机的时钟频率在一定程度上反映了机器速度。一般来讲,主频越高,速度越快。但是相同频率、不同体系结构的机器,速度可能会相差很多倍,因此还需要用其他方法来测定机器性能。

(2) 指令执行速度: 在计算机发展的初期,曾用加法指令的运算速度来衡量计算机的速度,速度是计算机的主要性能指标之一。因为加法指令的运算速度大体上反映出乘法、除法等其他算术运算的速度,而且逻辑运算、转移指令等简单指令的执行时间往往设计成与加法指令相同,因此加法指令的运算速度有一定代表性。当时,表征机器运算速度的单位是 KIPS(每秒千条指令),后来随着机器运算速度的提高,计量单位由 KIPS 发展到 MIPS(每秒百万条指令)。

(3) 等效指令速度法: 随着计算机指令系统的发展,指令的种类大大增加,用单种指令的 MIPS 值来表征机器的运算速度的局限性日益暴露,因此很快就出现了改进的办法,称之为吉普森(Gibson)混合法或等效指令速度法。

等效指令速度法统计各类指令在程序中所占比例,并进行折算。设某类指令 i 在程序中所占比例为 w_i , 执行时间为 t_i , 则等效指令的执行时间为:

$$T = \sum_{i=1}^n (w_i \times t_i)$$

其中 n 为指令的种类数。

(4) 数据处理速率(processing data rate, PDR)法: 因为在不同程序中,各类指令的使用频率是不同的,所以固定比例方法存在着很大的局限性;而且数据长度与指令功能的强弱对解题的速度影响极大。同时,这种方法也不能反映现代计算机中高速缓冲存储器(cache)、流水线、交叉存储等结构的影响。具有这种结构的计算机的性能不仅与指令的执行频率有关,而且也与指令的执行顺序与地址分布有关。

PDR 法采用计算 PDR 值的方法来衡量机器性能, PDR 值越大, 机器性能越好。PDR 与每条指令和每个操作数的平均位数以及每条指令的平均运算速度有关, 其计算方法如下:

$$PDR = L/R$$

其中: $L = 0.85G + 0.15H + 0.4J + 0.15K$

$$R = 0.85M + 0.09N + 0.06P$$

式中: G 是每条定点指令的位数; M 是平均定点加法时间;

H 是每条浮点指令的位数; N 是平均浮点加法时间;

J 是定点操作数的位数; P 是平均浮点乘法时间;

K 是浮点操作数的位数。

此外, 还作了如下规定: $G > 20$ 位, $H > 30$ 位; 从主存取一条指令的时间等于取一个字的时间; 指令与操作数存放在主存, 无变址或间址操作; 允许有并行或先行取指令功能, 此时选择平均取指令时间。PDR 值主要对 CPU 和主存储器的速度进行量度, 但不适合衡量机器的整体速度, 因为它没有涉及 cache、多功能部件等技术对性能的影响。

(5) 核心程序法: 上述性能评价方法主要针对 CPU (有时包括主存), 它没有考虑诸如 I/O 结构、操作系统、编译程序的效率等系统性能的影响, 因此难以准确评价计算机的实际工作能力。

核心程序法是研究较多的一种方法, 它把应用程序中用得最频繁的那部分核心程序作为评价计算机性能的标准程序, 在不同的机器上运行, 测得其执行时间, 作为各类机器性能评价的依据。机器软硬件结构的特点能在核心程序中得到反映, 但是核心程序各部分之间的联系较小。由于程序短, 所以访问存储器的局部性特征很明显, 以至于 cache 的命中率比一般程序高。

2. 基准测试程序

基准程序法 (benchmark) 是目前一致承认的测试性能的较好方法。有多种多样的基准程序, 如主要测试整数性能的基准程序、测试浮点性能的基准程序等。

(1) 整数测试程序: Dhrystone 是一个综合性的基准测试程序。它是为了测试编译器和 CPU 处理整数指令和控制功能的有效性, 人为地选择一些“典型指令”综合起来形成的测试程序。

用 C 语言编写的 Dhrystone 基准程序用了 100 条语句, 由下列操作组成: 各种赋值语句、各种数据类型和数据区、各种控制语句、过程调用和参数传递、整数运算和逻辑操作。

Dhrystone 程序测试的结果由每秒多少个 Dhrystones 来表示机器的性能。这个数值越大性能越好。VAX11/780 的测试结果为每秒 1757Dhrystones。为便于比较, 人们假设 1VAX MIPS = 每秒 1757Dhrystones, 将被测机器的结果除以 1757, 就得到被测机器相

对 VAX11/780 的 MIPS 值。有些厂家在宣布机器性能时就用 Dhrystone MIPS 值作为各自机器的 MIPS 值。

不过,不同的厂家在测试 MIPS 值时,使用的基准程序一般是不一样的。因此不同厂家机器的 MIPS 值有时虽然是相同的,但其性能却可能差别很大。那是因为各厂家在设计计算机时针对不同的应用领域(如科学和工程应用、商业管理应用、图形处理应用等)采用了不同的体系结构和实现方法。同一厂家的机器,采用相同的体系结构,用相同的基准程序测试,得到的 MIPS 值越大,一般说明机器速度越快。

(2) 浮点测试程序:在计算机科学和工程应用领域内,浮点计算工作量占很大比例,因此机器的浮点性能对系统的应用有很大的影响。有些机器只标出单个浮点操作性能,如浮点加法、浮点乘法时间。而大部分工作站则标出用 Linpack 和 Whetstone 基准程序测得的浮点性能。Linpack 主要测试向量性能和高速缓存性能。Whetstone 是一个综合性测试程序,除测试浮点操作外,还测试整数计算和功能调用等性能。

① 理论峰值浮点速度:巨型机和小型机在说明书中经常给出“理论峰值速度”的 MFLOPS 值。它不是机器实际执行程序时的速度,而是机器在理论上最大能完成的浮点处理速度。它不仅与处理机时钟周期有关,而且还与一个处理机里能并行执行操作的流水线功能部件数目和处理机的数目有关。多个 CPU 机器的峰值速度是单个 CPU 的峰值速度与 CPU 个数的乘积。

② Linpack 基准测试程序:Linpack 基准程序是一个用 FORTRAN 语言写成的子程序软件包,称为基本线性代数子程序包。此程序完成的主要操作是浮点加法和浮点乘法操作。测量计算机系统的 Linpack 性能时,让机器运行 Linpack 程序,测量运行时间,将结果用 MFLOPS 表示。

当解 n 阶线性代数方程组时, n 越大,向量化程度越高。其关系如表 1-6 所示。

向量化百分比指的是含向量成分的计算量占整个程序计算量的百分比。在同一台机器中,向量化程度越高,机器的运算速度越快,因为不管 n 的大小,求解方程时花在非向量操作的时间差不多是相等的。

③ Whetstone 基准测试程序:Whetstone 是用 FORTRAN 语言编写的综合性测试程序,主要由执行浮点运算、整数算术运算、功能调用、数组变址、条件转移和超越函数的程序组成。Whetstone 的测试结果用 kwips 表示,1kwips 表示机器每秒钟能执行 1000 条 Whetstone 指令。

表 1-6 矩阵的向量化程度

矩阵规模	100×100	300×300	1000×1000
向量化百分比	80%	95%	98%

(3) SPEC 基准程序(SPEC benchmark): SPEC 是 System Performance Evaluation Cooperation 的缩写,是由几十家世界知名的计算机大厂商所支持的非盈利的合作组织,旨在开发共同认可的标准基准程序。

SPEC 基准程序是由 SPEC 开发的一组用于计算机性能综合评价的程序。以对 VAX11/780 机的测试结果作为基数,其他计算机的测试结果以相对于这个基数的比率来表示。SPEC 基准程序能较全面地反映机器性能,有很高的参考价值。

SPEC 1.0 是 1989 年 10 月宣布的,是一套复杂的基准程序集,主要用于测量与工程和科学应用有关的数字密集型的整数和浮点数方面的计算。源程序超过 15 万行,包含 10 个测试程序,使用的数据量比较大,分别测试应用的各个方面。

SPEC 基准程序测试结果一般以 SPECmark(SPEC 分数)、SPECint(SPEC 整数)和 SPECfp(SPEC 浮点数)来表示。其中 SPEC 分数是 10 个程序的几何平均值;SPEC 整数是 4 个整数程序的几何平均值;SPEC 浮点数是 6 个浮点程序的几何平均值。

1992 年在原来 SPECint89 和 SPECfp89 的基础上又增加了两个整数测试程序和八个浮点数测试程序,因此 SPECint92 由 6 个程序组成,SPECfp92 由 14 个程序组成。这 20 个基准程序是基于不同的应用写成的,主要测 32 位 CPU、主存储器、编译器和操作系统的性能。

已有大约 30 个计算机软件和硬件公司参加了这个组织,它们都承认这种测试,并同意作为它们生产的机器或系统的测试标准。SPEC 基准程序的测试结果获得了普遍的认可。

参加这个组织的主要成员有: IBM, AT&T, BULL, CDC, DG, DEC, Fujitsu, HP, Intel, MIPS, Motorola, SGI, SUN 和 Unisys 等。1995 年这些厂商又共同推出了 SPECint95 和 SPECfp95 作为最新的测试标准程序。

(4) TPC 基准程序: TPC 是 Transaction Processing Council(事务处理委员会)的缩写。TPC 基准程序是由 TPC 开发的评价计算机事务处理性能的团试程序,用以评测计算机在事务处理、数据库处理、企业管理与决策支持系统等方面的性能。TPC 成立于 1988 年,目前已有 40 多个成员,几乎包括所有主要的商用计算机系统厂商和数据库厂商。TPC 分别于 1989 年 10 月、1990 年 8 月和 1992 年 7 月发表了 TPC-A, TPC-B 和 TPC-C 三个基准测试程序规范,计划在其后发表的有 TPC-D 和 TPC-E。该基准程序的评测结果用每秒完成的事务处理数 TPC 来表示。TPC 基准测试程序在商业界范围内建立了用于衡量机器性能以及性能价格比的标准。但是任何一种测试程序都有一定的适用范围,TPC 也不例外。

1.4.7 计算机故障诊断与容错

1. 计算机故障诊断技术

1) 计算机的故障

根据计算机故障表现出的特点,可以分为永久性、间歇性及瞬时性故障3类。

(1) 永久性故障:永久性故障表现出稳定性及持续性的特征,如元器件的损坏、电路的断线或短路、程序编写的错误等,它的特点是故障可以重复出现。

(2) 间歇性故障:间歇性故障表现出不稳定性及对系统状态具有依赖性的特征,此时可能表现出机器时好时坏的现象。

(3) 瞬时性故障:瞬时性故障是由偶然原因引起的短暂故障,一般无须修复就能恢复正常。但若频繁出现,也会影响工作,所以需要查出故障原因,以消除影响。

无论何种故障,均需及时发现,采取措施,避免故障影响的扩散。故障可能源于硬件,也可能由软件引起。

2) 故障诊断方法

故障诊断包括故障检测和故障定位两个方面。

故障检测:指测试并确定计算机系统有无故障的过程。

故障定位:指判定故障发生在某个子系统、功能块或器件的过程。

通常,故障诊断的主要方法有下述3种:

(1) 对电路直接进行测试的故障定位测试法:将被测试的系统划分成若干个测试域,并向这些域发送一系列调试码,然后收集并分析被调试区域的返回码,以确定故障位置或找出产生故障的元器件。

这种按线路内部的电路结构逐个进行逻辑关系测试的方法,将随着电路规模的增大而急骤地增加复杂性。因此,对计算机系统往往先采用下面提到的按功能测试的方法诊断出某一出故障的功能部件,然后再对这一部件的电路进行测试,或者干脆把这一产生故障的功能部件整个替换掉。

(2) “检查诊断程序”法:用机器语言写的“检查诊断程序”来进行诊断的方法是一种功能测试法。它利用机器指令的功能来对系统的某些部件进行测试。但由于一条指令的正确执行,往往涉及许多部件,因此故障定位所需的诊断时间较长,而且要求系统必须有能力保证诊断程序的正确执行,否则计算机连程序都不能运行,更谈不上诊断了。

(3) 微诊断法:在微程序控制的计算机中用微指令来对系统进行诊断称为微诊断法。由微指令组成的微诊断程序存放在控制存储器中或者先存放在外存储器中,诊断时再调入控制存储器。设计这种可写入的控制存储器称为动态微程序设计。微诊断法也是一种功能测试法。为了进行测试而必须保证工作只涉及较少部件,因此故障分辨得很细,

诊断程序运行的时间也较短。

不论采用指令或微指令进行诊断,都是采用“滚雪球”的方法。先对系统的某一部分进行测试,如果没有发现错误,就在这部分的基础上对其他未测试部分进行测试。如同“滚雪球”般,越滚越大,直到全部测试完成。

2. 计算机容错技术

容错技术指采用冗余方法来消除故障影响。针对硬件,有时间冗余和元器件冗余两种方法。

时间冗余:对同一计算进行重复运算,并对结果进行比较,或进行验算等,这种方法对解决偶然性故障比较有效。

元器件冗余,即利用附加的硬件来保证在局部有故障的情况下系统能正常工作。

容错的技术在不断进步,从最初的简单双机冗余到越来越高级的容错系统设计,保障了系统的可靠性和稳定性。下面介绍两种常用的容错技术。

(1) 简单的双机备份:在 20 世纪 60 年代主要采用双处理器或双机的方法来达到容错的目的。例如把关键的部件(处理器、存储器等)或整个计算机设置成两套:一套在系统运行时使用,另一套用作备份。根据系统的工作情况又可分为:热备份和冷备份两种。

① **热备份(双重系统):**两套系统同时同步运行,当联机子系统检测到错误时,退出服务进行检修,而由“热备份”子系统接替工作。

② **冷备份(双工系统):**处于冷备份的子系统平时停机或者运行与联机系统无关的运算,当联机子系统产生故障时,人工或自动进行切换,使冷备份系统成为联机系统。在冷备份时,不能保证从程序端点处精确地连续工作,因为备份机不能取得原来机器上当前运行的全部数据。

(2) 操作系统支持的双机容错:在 20 世纪 70 年代中期出现了软件和硬件结合的容错方法。该方法在操作系统的层次上,支持联机维修,即故障部分退出运行,进行维修并重新投入运行都不影响正在运行的应用程序。该结构的特点是系统内包括双处理器、双存储器、双输入输出控制器、不间断工作电源以及与之适应的操作系统等。因此,上述硬件的任一部分发生故障都不会影响系统的继续工作。系统容错是在操作系统控制下进行的,在每个处理机上都保持了反映所有系统资源状态的表格以及本机和他机的工作进程。

为了正常工作,必须保持双处理器工作同步,及时比较操作结果,并且提供检测机制以定位出故障的处理器(因为双机表决结果为 1:1,不能判定哪台机器出故障)。

综上所述,对故障处理步骤及方法归纳如下。

① **故障封闭:**防止故障扩散。

② **检错:**利用冗余代码,提供出错信息,例如奇偶校验码、汉明校验码、CRC 校验码等。

第2章 数据结构与算法

数据结构是计算机程序设计的重要理论和技术基础,它所涉及的内容和方法,无论是学习计算机领域的其他课程,还是对从事软件项目的开发都有着重要的作用。学习数据结构要达到的目标是:学会从问题入手,分析和研究计算机领域的数据结构特性,以便为应用所涉及的数据选择适当的逻辑结构、存储结构及其相应的操作算法,并初步掌握算法的时间复杂度和空间复杂度概念。数据抽象是一种创造性思维活动,这种能力是任何软件开发工具都无法取代的。

数据结构是指数据元素的集合(或数据对象)及元素间的相互关系和构造方法。例如,一个数据结构 B 可用一个二元组表示为: $B=(A,R)$,其中 A 是数据元素的非空有限集合, R 是定义在 A 上的关系的非空有限集合。结构就是元素之间的关系。在数据结构中,数据对象中元素之间的相互关系是数据的逻辑结构,数据元素及元素之间的关系的存储形式称为存储结构(或物理结构)。

数据结构按照逻辑关系的不同分为线性结构和非线性结构两大类,其中非线性结构又可分为树结构和图结构。

算法与数据结构密切相关。数据结构是算法设计的基础,算法总是建立在一定的数据结构基础之上,合理的数据结构可使算法简单而高效。

2.1 常用数据结构

2.1.1 线性表

1. 线性表

1) 线性表的定义

线性表是一类最简单、最常用的数据结构。简单来说,一个线性表是 n 个元素的有限序列,其中 $n \geq 0$,通常表示为 (a_1, a_2, \dots, a_n) 。其特点是,在非空的数据元素集合中:

- (1) 存在惟一的一个称作“第一个”的元素;
- (2) 存在惟一的一个称作“最后一个”的元素;
- (3) 除第一个元素外,集合中的每个元素均只有一个直接前驱;
- (4) 除最后一个元素外,集合中的每个元素均只有一个直接后继。

其中,(3)、(4)表现了线性表中元素之间的逻辑关系。

2) 线性表的存储结构

(1) 线性表的顺序存储: 线性表的顺序存储是用一组地址连续的存储单元依次存储线性表中的数据元素, 从而使得逻辑关系相邻的两个元素在物理位置上也相邻。在这种存储方式下, 存储逻辑关系无须占用额外的存储空间。其优点是可以随机存取表中的元素, 缺点是插入和删除操作需要移动大量的元素。

一般地, 在线性表的顺序存储结构中, 第 i 个元素 a_i 的存储位置为:

$$\text{LOC}(a_i) = \text{LOC}(a_1) + (i-1) \times L$$

其中, $\text{LOC}(a_1)$ 是表中第一个元素的存储位置, L 是表中每个元素所占空间的大小。根据这个计算关系, 可随机存取表中的任一个元素。

(2) 线性表的链式存储: 线性表的链式存储是用结点来存储数据元素。结点的空间可以是连续的, 也可以是不连续的, 因此存储数据元素的同时必须存储元素之间的逻辑关系。结点空间只有在需要的时候才申请, 无须事先分配。最基本的结点结构如下所示:

数据域	指针域
-----	-----

其中, 数据域用于存储数据元素的值, 指针域则存储当前元素的直接前驱或直接后继信息, 指针域中的信息称为指针(或链)。 n 个结点通过指针连成一个链表, 若结点中只有一个指针域, 则称为线性链表(或单链表)。

线性表采用链表作为存储结构时, 不能进行数据元素的随机访问, 其优点是插入和删除操作不需要移动元素。根据结点中指针信息的实现方式, 还有其他几种链表结构。

- 双向链表: 每个结点包含两个指针, 指明直接前驱和直接后继元素。可在两个方向上遍历其后及其前的元素。
- 循环链表: 表尾结点的指针指向表中的第一个结点, 可在任何位置上方向不变地遍历整个链表。
- 静态链表: 借助数组来描述线性表的链式存储结构。

在链式存储结构中, 只需要一个指针(头指针)指向第一个结点, 就可以顺序访问到表中的任意一个元素。为了简化对链表状态的判定和处理, 特别引入一个不存储数据元素的结点, 称为头结点, 将其作为链表的第一个结点并令头指针指向该结点。

3) 线性表的插入和删除运算

(1) 基于顺序存储结构的运算。

插入元素前要移动元素以挪出空的存储单元, 然后再插入元素。删除元素时同样需要移动元素, 以填充被删元素空出来的存储单元。在等概率下平均移动元素的次数分

别为：

$$E_{\text{insert}} = \sum_{i=1}^{n-1} P_i \times (n-i+1) = \frac{1}{n+1} \sum_{i=1}^{n+1} (n-i+1) = \frac{n}{2}$$

$$E_{\text{delete}} = \sum_{i=1}^n q_i \times (n-i) = \frac{1}{n} \sum_{i=1}^n (n-i) = \frac{n-1}{2}$$

(2) 基于链式存储结构的运算。

在链式存储结构下,进行插入和删除,其实质都是对相关指针的修改。

- 在单向链表中插入结点时,指针的变化情况如图 2-1 所示。
- 在单向链表中删除结点时,指针的变化情况如图 2-2 所示。

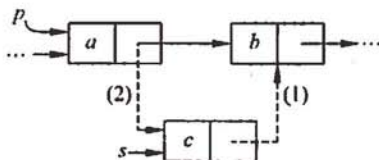


图 2-1 在单向链表中插入结点时的指针变化示意图

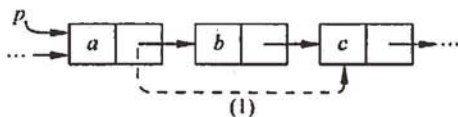


图 2-2 在单向链表中删除结点时的指针变化示意图

- 在双向链表中插入结点时,指针的变化情况如图 2-3 所示。
- 在双向链表中删除结点时,指针的变化情况如图 2-4 所示。

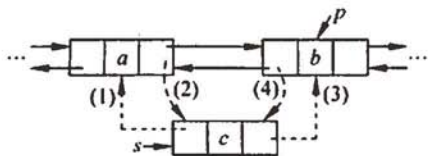


图 2-3 双向链表插入结点时的指针变化示意图

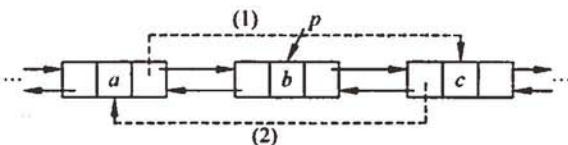


图 2-4 双向链表删除结点时的指针变化示意图

2. 栈和队列

1) 栈的定义及基本运算

(1) 栈的定义。

栈是只能通过访问它的一端来实现数据存储和检索的一种线性数据结构。换句话

说,栈的修改是按先进后出的原则进行的。因此,栈又称为先进后出(FILO,或后进先出)的线性表。对栈来说,进行插入和删除操作的一端称为栈顶(top),相应地,另一端称为栈底(bottom)。不含数据元素的栈称为空栈。

(2) 栈的基本运算。

- 置空栈 InitStack(S): 创建一个空栈 S。
- 判栈空 Empty(S): 当栈 S 为空栈时返回“真”值,否则返回“假”值。
- 入栈 Push(S,x): 将元素 x 加入栈顶,并更新栈顶指针。
- 出栈 Pop(S): 将栈顶元素从栈中删除,并更新栈顶指针。若需要得到栈顶元素的值,可将 Pop(S) 定义为一个函数,返回栈顶元素的值。
- 读栈顶元素 Top(S): 返回栈顶元素的值,但不修改栈顶指针。

利用上述 5 种基本运算,可以实现基于栈结构的应用问题的求解。

2) 栈的存储结构

(1) 栈的顺序存储。

栈的顺序存储指用一组地址连续的存储单元依次存储自栈顶到栈底的数据元素,同时附设指针 top 指示栈顶元素的位置。采用顺序存储结构的栈也称为顺序栈。在顺序存储方式下,需要预先定义或申请栈的存储空间,也就是说,栈空间的容量是有限的。因此在顺序栈中,当一个元素入栈时,需要判断是否栈满(栈空间中没有空闲单元)。若栈满,则元素入栈会发生上溢现象。

利用栈底位置相对不变的特性,可让两个顺序栈共享一个一维数据空间,以互补余缺,实现方法是:将两个栈的栈底位置分别设在存储空间的两端,让它们的栈顶各自向中间延伸,如图 2-5 所示。这样,两个栈的空间可以相互调节,只有在整个存储空间被占满时才发生上溢,产生上溢的概率要小得多。

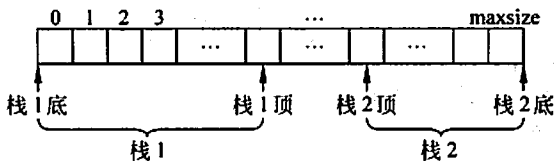


图 2-5 两个栈共享空间示意图

(2) 栈的链式存储。

为了克服顺序存储结构的栈可能存在上溢的不足,可以用链式存储结构实现栈中元素的存储。用链表作为存储结构的栈也称为链栈。由于栈中元素的插入和删除仅在栈顶一端进行,因此不必设置头结点,链表的头指针就是栈顶指针。链栈的表示如图 2-6 所示。

3) 队列的定义及基本运算

(1) 队列的定义。

队列是一种先进先出(FIFO)的线性表,它只允许在表的一端插入元素,而在表的另一端删除元素。在队列中,允许插入元素的一端称为队尾(rear),允许删除元素的一端称为队头(front)。

(2) 队列的基本运算。

- 置队空 InitQueue(Q): 创建一个空的队列 Q。
- 判队空 Empty(Q): 当队列为空时返回“真”值,否则返回“假”值。
- 入队 EnQueue(Q, x): 将元素 x 加到队列 Q 的队尾,并更新队尾指针。
- 出队 DeQueue(Q): 将队头元素从队列 Q 中删除,并更新队头指针。
- 读队头元素 Frontque(Q): 返回队头元素的值,但不更新队头指针。

同栈结构一样,将队列的五种基本运算进行组合,就可以完成队列的各种运算。

4) 队列的存储结构

(1) 队列的顺序存储。

队列的顺序存储结构又称为顺序队列,它也是利用一组地址连续的存储单元存放队列中的元素。由于队列中元素的插入和删除限定在表的两端进行,因此需要设置队头指针和队尾指针,分别指示出当前的队首元素和队尾元素。

设顺序队列 Q 的容量为 6,其队头指针为 front,队尾指针为 rear,头、尾指针和队列中元素之间的关系如图 2-7 所示。

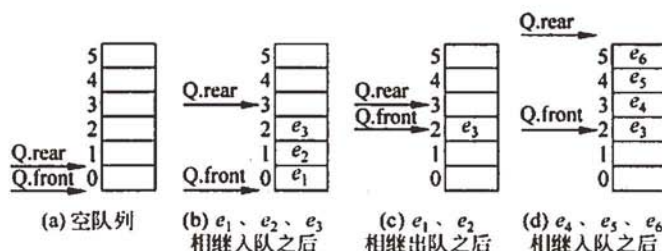


图 2-7 队列的头、尾指针与队列中元素之间的关系

在顺序队列中,为了降低运算的复杂度,元素入队时,只需修改队尾指针,元素出队时,只须修改队头指针。由于顺序队列的存储空间是提前设定的,所以队尾指针会有一个上限值,当队尾指针达到其上限时,就不能只通过修改队尾指针来实现新元素的入队操作了,如图 2-8 所示。此时,可通过整除取余运算将顺序队列假想成一个环状结构,称之为循环队列。从图 2-8 中可以看出,在队列空和队列满的情况下,循环队列的队头、队尾指

针指向的位置是相同的。为了区别队空和队满的情况,可采用两种处理方式:其一是设置一个标志位,以区别头、尾指针的值相同时队列是空还是满;其二是牺牲一个元素空间,约定以“队列的尾指针所指位置的下一个位置是头指针时”表示队列满,头、尾指针的值相同时表示队列为空。

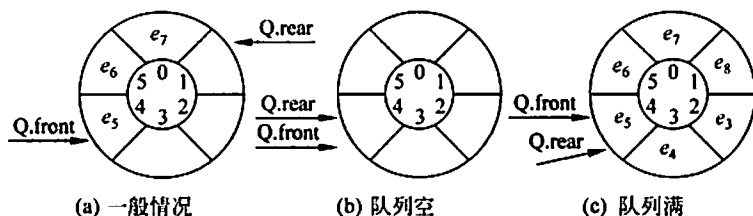


图 2-8 循环队列的头、尾指针示意图

(2) 队列的链式存储。

队列的链式存储也称为链队列。这里为了便于操作,给链队列添加一个头结点,并令头指针指向头结点。因此,队列为空的判定条件是:头指针和尾指针的值相同,且均指向头结点。队列的链式存储结构如图 2-9 所示。

队列结构常用于处理需要排队的场合,如操作系统中处理打印任务的打印队列,以及离散事件的计算机模拟等。

3. 串

1) 串的定义及基本运算

(1) 串的定义。

串是仅由字符构成的有限序列,是取值范围受限的线性表。一般记为 $S = 'a_1 a_2 \dots a_n'$, 其中 S 是串名,单引号括起来的字符序列是串值。

(2) 串的几个基本概念。

- 空串: 长度为零的串,空串不包含任何字符。
- 空格串: 由一个或多个空格组成的串。虽然空格是一个空白符,但它也是一个字符,计算串长度时要将其计算在内。
- 子串: 由串中任意长度的连续字符构成的序列称为子串,含有子串的串称为主串。子串在主串中的位置指子串首次出现时,该子串的第一个字符在主串的位置。空串是任意串的子串。
- 串相等: 指两个串长度相等且对应位置上的字符也相同。
- 串比较: 两个串比较大小时以字符的 ASCII 码值作为依据。比较操作从两个串的第一个字符开始进行,字符的 ASCII 码值大者所在的串为大。若其中一个串先

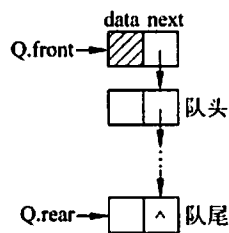


图 2-9 链队列示意图

结束,则以串长较大者为大。

(3) 串的基本操作。

- 赋值操作 `StrAssign(s,t)`: 将串 `t` 的值赋给串 `s`。
- 连接操作 `Concat(s,t)`: 将串 `t` 接续在串 `s` 的尾部,形成一个新串。
- 求串长 `StrLength(s)`: 返回串 `s` 的长度。
- 串比较 `StrCompare(s,t)`: 比较两个串的大小。返回值 `-1`、`0` 和 `1`, 分别表示 $s < t$ 、 $s = t$ 和 $s > t$ 3 种情况。
- 求子串 `SubString(s,start,len)`: 返回串 `s` 中从 `start` 开始的、长度为 `len` 的字符序列。

以上 5 种最基本的串操作构成了串的最小操作子集,利用它们就可以实现串的其他运算。

2) 串的存储结构

(1) 串的静态存储: 定长存储结构。

串的静态存储结构就是串的顺序存储结构,用一组地址连续的存储单元来存储串值的字符序列。由于串中的元素为字符,所以可通过程序语言提供的字符数组定义串的存储空间,也可以根据串长的需要动态申请字符串的空间。

(2) 串的链式存储: 块链。

字符串也可以采用链表作为存储结构,当用链表存储串中的字符时,每个结点中可以存储一个字符,也可以存储多个字符,但要考虑存储密度问题。结点大小为 4 的块链如图 2-10 所示。

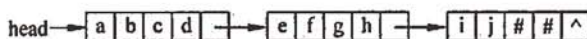


图 2-10 串值的链表存储方式

在链式存储结构中,结点大小的选择和顺序存储方法中数组空间大小的选择一样重要,它直接影响对串处理的效率。

3) 串的模式匹配

子串的定位操作通常称为串的模式匹配,它是各种串处理系统中最重要、最基本的运算之一。子串也称为模式串。

(1) 朴素的模式匹配算法。

该算法也称为布鲁特—福斯算法,其基本思想是从主串的第一个字符起与模式串的第一个字符比较。若相等,则继续对后续的逐个字符进行比较。否则从主串的第二个字符起与模式串的第一个字符重新比较,直至模式串中每个字符依次和主串中的一个连续的字符序列相等时为止。此时称为匹配成功,否则称为匹配失败。

假设主串的长度为 n , 模式串的长度为 m , 下面分析朴素模式匹配算法的时间复杂度。设从主串的第 i 个位置开始与模式串匹配成功, 而在前 $i-1$ 趟匹配中, 每趟不成功的匹配都是模式串的第一个字符与主串中相应的字符不相同, 则在前 $i-1$ 趟匹配中, 字符的比较共进行了 $i-1$ 次。因第 i 趟成功匹配的字符比较次数为 m , 所以总的字符比较次数为 $(i-1+m)$ 且 $1 \leq i \leq n-m+1$ 。若在这 $n-m+1$ 个起始位置上匹配成功的概率相同, 则在最好情况下, 匹配成功时字符间的平均比较次数为:

$$\sum_{i=1}^{n-m+1} p_i (i-1+m) = \frac{1}{n-m+1} \sum_{i=1}^{n-m+1} (i-1+m) = \frac{1}{2}(n+m)$$

因此, 在最好情况下匹配算法的时间复杂度为 $O(n+m)$ 。

而在最坏的情况下, 每一趟不成功的匹配都是模式串的最后一个字符与主串中相应的字符不相等, 则主串中新一趟的起始位置为 $i-m+2$ 。若设第 i 趟匹配时成功, 则在前 $i-1$ 趟不成功的匹配中, 每趟都比较了 m 次, 总共比较了 $i \times m$ 次。因此, 最坏情况下的平均比较次数为:

$$\sum_{i=1}^{n-m+1} p_i (i \times m) = \frac{m}{n-m+1} \sum_{i=1}^{n-m+1} i = \frac{1}{2} m(n+m)$$

由于 $n \gg m$, 所以该算法在最坏情况下的时间复杂度为 $O(n \times m)$ 。

(2) 改进的模式匹配算法。

改进的模式匹配算法又称为 KMP 算法。其改进之处在于: 每当匹配过程中出现相比较的字符不相等时, 不需要回溯主串的指针, 而是利用已经得到的“部分匹配”的结果, 将模式串向后“滑动”尽可能远的距离, 再继续进行比较。此算法可在 $O(n+m)$ 的时间内完成。

设模式串为 $'p_1 p_2 \cdots p_m'$, 在 KMP 算法中, 依据模式串的 next 函数值实现子串的滑动。若令 $\text{next}[j]=k$, 则 $\text{next}[j]$ 表示当模式串中的 p_j 与主串中相应字符不相等时, 令模式串的 p_k 与主串的相应字符进行比较, 从而使主串的字符位置指针无须回溯。next 函数的定义如下:

$$\text{next}[j] = \begin{cases} 0 & \text{当 } j=1 \text{ 时} \\ \max\{k \mid 1 < k < j \text{ 且 } 'p_1 p_2 \cdots p_{k-1}' = 'p_{j-k+1} p_{j-k+2}, \dots, p_{j-1}'\} & \\ 1 & \text{其他情况} \end{cases}$$

2.1.2 数组、矩阵和广义表

1. 数组

1) 数组的定义及基本运算

(1) 数组的定义。

n 维数组是一种“同构”的数据结构,其每个数据元素类型相同,结构一致。数组是定长线性表在维数上的扩张,即线性表中的元素又是一个线性表。

设有 n 维数组 $a[b_1, b_2, \dots, b_n]$, 其每一维的下界都为 1, b_i 是第 i 维的上界。从数据结构的逻辑关系角度来看, a 中的每个元素 $a[j_1, j_2, \dots, j_n]$ ($1 \leq j_i \leq b_i$) 都被 n 个关系所约束。在每个关系中, 除第一个和最后一个元素外, 其余元素都只有一个直接后继和一个直接前驱。因此就单个关系而言, 这 n 个关系仍是线性的。

以二维数组 $A[m][n]$ 为例, 可以把它看成一个定长的线性表, 它的每个元素也是一个定长线性表。

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

A 可看成一个行向量形式的线性表, 如下所示:

$$A_{\text{row}} = [[a_{11} a_{12} \cdots a_{1n}], [a_{21} a_{22} \cdots a_{2n}], \dots, [a_{m1} a_{m2} \cdots a_{mn}]];$$

或列向量形式的线性表:

$$A_{\text{col}} = [[a_{11} a_{21} \cdots a_{m1}], [a_{12} a_{22} \cdots a_{m2}], \dots, [a_{1n} a_{2n} \cdots a_{mn}]]$$

数组结构的特点是:

- 数据元素数目固定。一旦定义了一个数组结构, 就不再有元素的增减变化;
- 数据元素具有相同的类型;
- 数据元素的下标关系受上下界的约束且下标有序。

(2) 数组的两个基本运算。

- 给定一组下标, 存取相应的数据元素;
- 给定一组下标, 修改相应的数据元素中某个数据项的值。

几乎所有的计算机程序设计语言都提供了数组类型。实际上, 在语言中把数组看成具有共同名字的同类型多个变量的集合。需要注意的是, 不能对数组进行整体的运算, 只能对单个数组元素进行运算。

2) 数组的顺序存储

由于数组一般不作插入和删除运算, 也就是说, 一旦定义了数组, 则结构中的数据元素个数和元素之间的关系就不再发生变动, 因此数组适合于采用顺序存储结构。

由于计算机的内存结构是一维线性的, 因此存储多维数组时必须按某种方式进行降维处理, 即将数组元素排成一个线性序列, 这就产生了次序约定问题。因为多维数组是由较低一维的数组定义的, 依次类推, 就可通过这种递推关系将多维数组的数据元素排成一个线性序列。

对于数组,一旦确定了它的维数和各维的长度,便可为它分配存储空间。反之,只要给出一组下标便可求得相应数组元素的存储位置,也就是说,在数据的顺序存储结构中,数据元素的位置是其下标的线性函数。

二维数组的存储结构可分为以行为主序和以列为主序的两种方法,如图 2-11 所示。

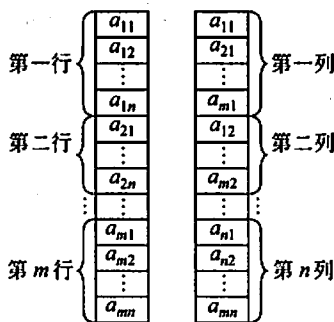


图 2-11 二维数组的两种存储方式

设每个数据元素占用 L 个单元, m, n 为数组的行数和列数,那么以行为主序优先存储的地址计算公式为:

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + ((i-1) \times n + (j-1)) \times L$$

同样,以列为主序优先存储的地址计算公式为:

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + ((j-1) \times m + (i-1)) \times L$$

推广至多维数组的存储,按下标顺序存储表示先排最右的下标,从右向左直至排到最左下标,而逆下标顺序则正好相反。

2. 矩阵

矩阵是很多科学与工程计算问题中研究的数学对象。在数据结构中,我们感兴趣的是如何存储矩阵中的元素,从而使矩阵的各种运算能有效地进行。

在一些矩阵中,存在很多值相同的元素或者是零元素。为了节省存储空间,可以对这类矩阵进行压缩存储。压缩存储的含义是为多个值相同的元素只分配一个存储单元,对零元不分配存储单元。对这类矩阵而言,假如值相同的元素或零元在矩阵中的分布有一定的规律,则称此类矩阵为特殊矩阵,否则称为稀疏矩阵。

1) 特殊矩阵

若矩阵中元素(或非零元素)的分布有一定的规律,则称之为特殊矩阵。常见的特殊矩阵有对称矩阵、三角矩阵和对角矩阵等。对于特殊矩阵,由于其非零元的分布都有一定的规律,所以可将其压缩存储在一维数组中,并建立起每个非零元在矩阵中的位置与其在

一维数组中的位置之间的对应关系。

若矩阵 $A_{n \times n}$ 中的元素有以下特点：

$$a_{ij} = a_{ji} \quad 1 \leq i, j \leq n$$

则称之为 n 阶对称矩阵。

若为对称矩阵中的每一对元素分配一个存储单元,那么就可将 n^2 个元素压缩存储到能存放 $n(n+1)/2$ 个元素的存储空间中。为不失一般性,以行为主序存储下三角(包括对角线)中的元素。假设以一维数组 $B[n(n+1)/2]$ 作为 n 阶对称矩阵 A 的存储结构,则 $B[k]$ ($0 \leq k < n(n+1)/2$) 与矩阵元素 a_{ij} 之间存在着——对应的关系:

$$k = \begin{cases} \frac{j(i-1)}{2} + j - 1 & \text{当 } i \geq j \\ \frac{j(j-1)}{2} + i - 1 & \text{当 } i < j \end{cases}$$

对角矩阵是指矩阵中的非零元素都集中在以主对角线为中心的带状区域中,即除了主对角线上和直接在对角线上、下方若干条对角线上的元素外,其余的矩阵元素都为零。一个 n 阶的三对角矩阵如图 2-12 所示。

$$A_{n \times n} = \begin{bmatrix} a_{1,1} & a_{1,2} & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & 0 \\ & a_{3,2} & a_{3,3} & a_{3,4} & & \\ & & \dots & \dots & \dots & \\ & & & a_{i,i-1} & a_{i,i} & a_{i,i+1} \\ & 0 & & \dots & \dots & \dots \\ & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

图 2-12 三对角矩阵示意图

若以行为主序将 n 阶三对角矩阵 $A_{n \times n}$ 的非零元素存储在一维数组 $B[k]$ ($0 \leq k < 3 \times n - 2$) 中,则元素位置之间的对应关系为:

$$k = 3 \times (i-1) - 1 + j - i + 1 = 2i + j - 3 \quad (1 \leq i, j \leq n)$$

其他特殊矩阵可作类似的计算,这里不再一一说明。

2) 稀疏矩阵

在一个矩阵中,若非零元素的个数远远少于零元素的个数,且非零元素的分布没有规律,则称之为稀疏矩阵。对于稀疏矩阵,存储非零元素时必须同时存储其位置(即行号和列号),所以三元组 (i, j, a_{ij}) 可惟一地确定矩阵 A 中的一个元素。由此,一个稀疏矩阵可由表示非零元素的三元组及其行、列数惟一确定。图 2-13 所示的是一个六行七列的稀疏

矩阵,其三元组表为:

$((1,2,12),(1,3,9),(3,1,-3),(3,6,14),(4,3,24),(5,2,18),(6,1,15),(6,4,-7))$ 。

$$M_{6 \times 7} = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

图 2-13 稀疏矩阵示意图

稀疏矩阵的三元组表的顺序存储结构称为三元组顺序表,而常用的三元组表的链式存储结构是十字链表。

3. 广义表

1) 广义表的定义

广义表是线性表的推广,是由零个或多个单元素或子表所组成的有限序列。

广义表与线性表的区别在于:线性表的元素都是结构上不可分的单元素,而广义表的元素既可以是单元素,也可以是有结构的表。广义表一般记为:

$$LS = (a_1, a_2, \dots, a_n)$$

其中 $a_i (1 \leq i \leq n)$ 既可以是单个元素,又可以是广义表,分别称为原子和子表。

广义表的长度是指广义表中元素的个数。广义表的深度是指广义表展开后所含的括号的最大层数。

2) 广义表的基本操作

与线性表类似,广义表也有查找、插入和删除等操作。由于广义表的结构较复杂,其各种运算的实现也不如线性表简单,这里只讨论两个重要的运算:

- 取表头 $head(LS)$: 非空广义表 LS 的第一个元素称为表头,它可以是一个单元素,也可以是一个子表。
- 取表尾 $tail(LS)$: 在非空广义表中,除表头元素之外,由其余元素构成的表称为表尾。非空广义表的表尾必定是一个表。

3) 广义表的特点

- 广义表可以是多层次的结构,因为广义表的元素可以是子表,而子表的元素还可以是子表;
- 广义表中的元素可以是已经定义的广义表的名字,所以一个广义表可为其他广义表共享;
- 广义表可以是一个递归的表,即广义表中的元素也可以是本广义表的名字。

4) 广义表的存储结构

由于广义表中的元素本身还可以具有结构,它是一种带有层次的非线性结构,因此难以用顺序存储结构表示,通常采用链式存储结构存储广义表。由上面的讨论可知,若广义表不空,则可分解为表头和表尾两部分。反之,一对确定的表头和表尾可惟一地确定一个广义表。针对原子和子表可分别设计不同的结点结构,如图 2-14 所示。

对于广义表 $LS=(a,(b,c,d))$,其链式存储结构如图 2-15 所示。



图 2-14 广义表的链表结点结构

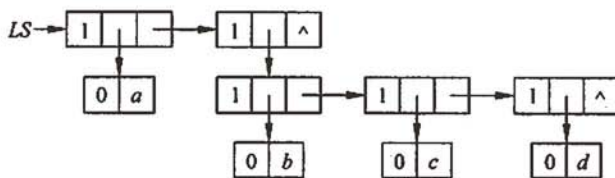


图 2-15 广义表的存储结构示意图

2.1.3 树

1. 树的定义及基本运算

1) 树的定义

树是 $n (n \geq 0)$ 个结点的有限集合。当 $n=0$ 时称为空树。在任一非空树 ($n > 0$) 中满足①有且仅有一个称为根的结点；②其余结点可分为 $m (m \geq 0)$ 个互不相交的有限集 T_1, T_2, \dots, T_m , 其中每个集合又都是一棵树,并且称为根结点的子树。

树的定义是递归的,它表明了树本身的固有特性,也就是说,一棵树由若干棵子树构成,而子树又由更小的子树构成。

该定义只给出了树的组成特点,若从数据结构的逻辑关系角度来看,树中元素之间有明显的层次关系。对树中的某个结点而言,它最多只和上一层的一个结点(即其双亲结点)有直接的关系,而与其下一层的多个结点(即其子树结点)有直接关系。通常,凡是分等级的分类方案都可以用具有严格层次关系的树结构来描述。

2) 树中的基本概念

- 双亲和孩子: 对于结点的子树而言,其根称为该结点的孩子。相应地,该结点称

为其子结点的双亲。

- 兄弟：具有相同双亲的结点互为兄弟。
- 结点的度：一个结点的子树的个数记为该结点的度。
- 叶子结点：也称为终端结点，指度为零的结点。
- 内部结点：度不为零的结点称为分支结点或非终端结点。除根结点之外，分支结点也称为内部结点。
- 结点的层次：根为第 1 层，根的孩子为第 2 层。依此类推，若某结点在第 i 层，则其孩子结点就在第 $i+1$ 层。
- 树的高度：一棵树的最大层次数记为树的高度（或深度）。
- 有序（无序）树：若树中结点的各子树从左到右具有次序，即不能交换，则称该树为有序树，否则称为无序树。
- 森林： m ($m \geq 0$) 棵互不相交的树的集合。

3) 树的遍历运算

在应用树结构时，常要求按某种次序获得树中全部结点的信息，这可通过树的遍历操作来实现。树的遍历操作也是树中其他运算的重要基础。

2. 二叉树的定义及基本运算

1) 二叉树的定义

二叉树是 n ($n \geq 0$) 个结点的有限集合。它或者是空树 ($n=0$)，或者是由一个根结点及两棵不相交的分别称为左、右子树的二叉树所组成。可见二叉树同样具有递归性质。

特别需要注意的是，尽管普通树和二叉树的概念之间有许多联系，但它们是有许多不同。普通树和二叉树之间最主要的区别是：二叉树的结点的子树要区分左子树和右子树，即使在结点只有一棵子树的情况下也要明确指出该子树是左子树还是右子树；另外，二叉树的结点的最大度为 2，而普通树中不限制结点的度数。图 2-16 所示的两棵不同的二叉树虽然与图 2-17 所示的普通树相似，但却不等同于普通树。

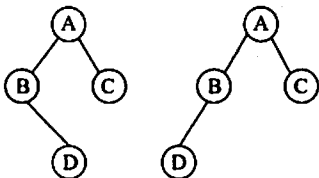


图 2-16 两棵不同的二叉树

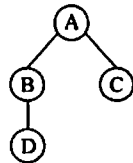


图 2-17 一棵普通树

2) 二叉树的运算

二叉树的基本运算是遍历，其他运算可建立在遍历运算的基础上。

3. 二叉树的性质

(1) 二叉树第 i 层 ($i \geq 1$) 上至多有 2^{i-1} 个结点。

只要对层数 i 采用数学归纳法即可证明此性质成立。

(2) 深度为 k 的二叉树至多有 $2^k - 1$ 个结点 ($k \geq 1$)。

(3) 对任何一棵二叉树, 若其终端结点数为 n_0 , 度为 2 的结点数为 n_2 , 则 $n_0 = n_2 + 1$ 。

(4) 具有 n 个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$ 。

(5) 对一棵有 n 个结点的完全二叉树的结点按层次自左至右进行编号, 则对任一结点 i ($1 \leq i \leq n$) 有:

- 若 $i=1$, 则结点 i 是二叉树的根, 无双亲; 若 $i>1$, 则其双亲为 $\lfloor i/2 \rfloor$ 。
- 若 $2i > n$, 则结点 i 无左孩子, 否则其左孩子为 $2i$ 。
- 若 $2i+1 > n$, 则结点 i 无右孩子, 否则其右孩子为 $2i+1$ 。

若深度为 k 的二叉树有 $2^k - 1$ 个结点, 则称其为满二叉树。可以对满二叉树中的结点进行连续编号, 约定编号从根结点起, 自上而下、自左至右依次进行。深度为 k 、有 n 个结点的二叉树, 当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时, 称之为完全二叉树。满二叉树和完全二叉树如图 2-18 所示。

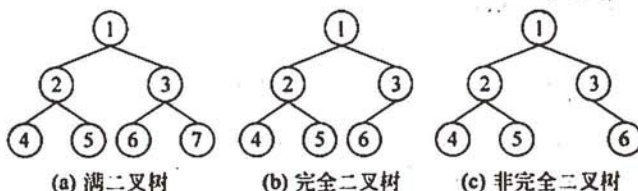


图 2-18 满二叉树和完全二叉树示意图

4. 二叉树的存储结构

1) 二叉树的顺序存储结构

用一组地址连续的存储单元存储二叉树中的数据元素, 必须把结点排成一个适当的线性序列, 并且结点在这个序列中的相互位置能反映出结点之间的逻辑关系。对于深度为 k 的完全二叉树, 除第 k 层外, 其余各层中含有最大的结点数, 即每一层的结点数恰为其上一层结点的两倍, 由此从一个结点的编号可推知其双亲、左孩子和右孩子的编号。

对于编号为 i 的结点, 则有:

- 若 $i=1$ 时, 该结点为根结点, 无双亲;
- 若 $i>1$ 时, 该结点的双亲结点为 $\lfloor i/2 \rfloor$;
- 若 $2i \leq n$, 则该结点的左孩子编号为 $2i$, 否则无左孩子;
- 若 $2i+1 \leq n$, 则该结点的右孩子编号为 $2i+1$, 否则无右孩子;

- 若 i 为奇数且不为 1, 则该结点左兄弟的编号为 $i-1$, 否则无左兄弟;
- 若 i 为偶数且小于 n , 则该结点右兄弟的编号为 $i+1$, 否则无右兄弟。

完全二叉树的顺序存储结构如图 2-19 (a) 所示。

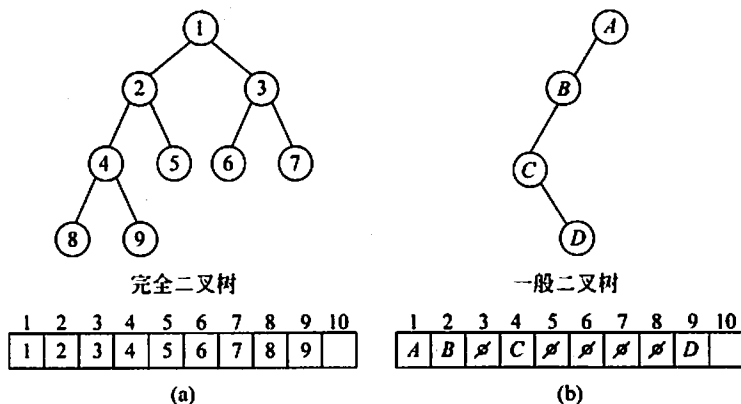


图 2-19 二叉树的顺序存储结构

显然顺序存储结构对完全二叉树而言既简单又节省空间,但对于一般二叉树则不适用。因为在顺序存储结构中,是以结点在存储单元中的位置来表示结点之间的关系的,那么对于一般的二叉树来说,也必须按照完全二叉树的形式存储,也就是要添上一些实际并不存在的“虚结点”,这将造成空间的浪费,如图 2-18(b)所示。

在最坏的情况下,一个深度为 k 且只有 k 个结点的二叉树(单支树)却需要 $2^k - 1$ 个存储单元。

2) 二叉树的链式存储结构

由于二叉树中结点含有数据元素、左子树根、右子树根及双亲等信息,因此可以用三叉链表或二叉链表(即一个结点含有 3 个指针或含有两个指针)来存储二叉树,链表的头指针指向二叉树的根结点,如图 2-20 所示。

在不同的存储结构中,实现二叉树的算法亦不同,具体应采用什么存储结构,除考虑二叉树的形态外还应考虑需要进行的运算。

5. 二叉树的遍历

遍历是按某种策略访问树中的每个结点,且仅访问一次。

由于二叉树所具有的递归性质,一棵非空的二叉树可以看作由根结点、左子树和右子树 3 部分构成。因此,若能依次遍历这 3 部分的信息,也就遍历了整棵二叉树。按照遍历左子树要在遍历右子树之前进行的原则,根据访问根结点位置的不同,可得到二叉树的前序(或称先序)、中序和后序 3 种遍历方法。

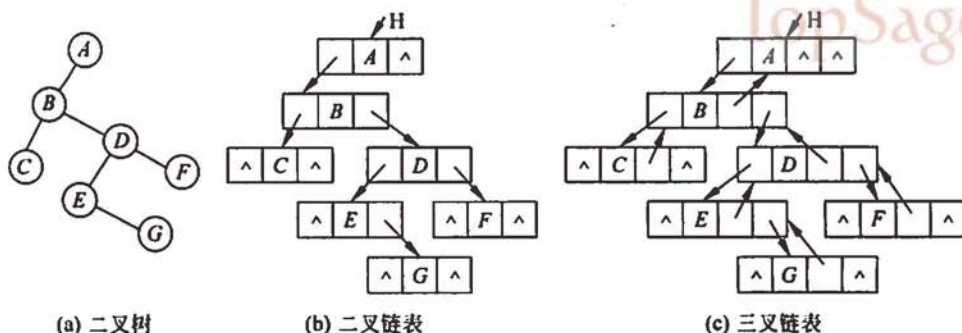


图 2-20 二叉树的链表存储结构

(1) 先序遍历二叉树的操作定义为：

若二叉树为空，则进行空操作；否则：

- 访问根结点；
- 先序遍历根的左子树；
- 先序遍历根的右子树。

(2) 中序遍历二叉树的操作定义为：

若二叉树为空，则进行空操作；否则：

- 中序遍历根的左子树；
- 访问根结点；
- 中序遍历根的右子树。

(3) 后序遍历二叉树的操作定义为：

若二叉树为空，则进行空操作；否则：

- 后序遍历根的左子树；
- 后序遍历根的右子树；
- 访问根结点。

从树的根结点出发，3 种方法的遍历路线如

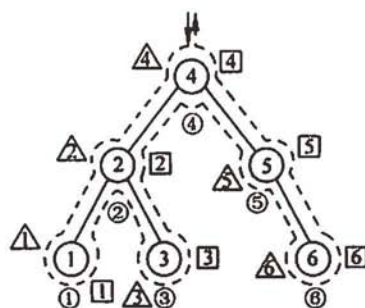


图 2-21 3 种遍历过程执行示意图

图 2-21 所示。该路线从根结点出发，逆时针沿着

二叉树的外缘移动，对每个结点均途经 3 次。若第一次经过结点时即访问，则是先序遍历；若第二次（或第三次）经过结点时访问结点，则是中序遍历（或后序遍历）。因此，只要将搜索路线上所有在第一次、第二次和第三次经过的结点信息分别输出，即可分别得到该二叉树的先序、中序和后序遍历序列。粗略地讲，若去掉 3 种遍历方法中的访问操作，则 3 种遍历方法基本相同。这说明 3 种遍历的搜索途径相同。

遍历二叉树的基本操作就是访问结点。不论按照哪种次序遍历，对含有 n 个结点的

二叉树,遍历算法的时间复杂度都为 $O(n)$ 。因为在遍历的过程中,每进行一次递归调用,都是将函数的“活动记录”压入栈中。因此,栈的最大高度恰为树的深度。所以,在最坏情况下,二叉树是有 n 个结点且深度为 n 的单枝树,遍历算法的空间复杂度也为 $O(n)$ 。

遍历二叉树的过程实质上是按一定规则,将树中的结点排成一个线性序列的过程,因此遍历操作得到的是树中结点的一个线性序列。在每一种序列中,有且仅有一个起始点和一个终结点,其余结点有且仅有惟一的直接前驱和直接后继。显然,关于结点的前驱和后继的讨论是针对某一个遍历序列而言的。

对二叉树还可以进行层序遍历。设二叉树的根结点所在层数为 1,层序遍历就是从树的根结点出发,首先访问第 1 层的树根结点,然后从左到右依次访问第 2 层上的结点,接着是第 3 层上的结点,依此类推,自上而下、自左至右逐层访问树中各层结点的过程就是层序遍历。

6. 线索二叉树

1) 引入线索二叉树

二叉树的遍历实质上是对一个非线性结构实现线性化的过程,使每个结点(除第一个和最后一个外)在这些线性序列中有且仅有一个直接前驱和直接后继。但在二叉链表存储结构中,只能找到一个结点的左、右孩子信息,而不能直接得到结点在任一遍历序列中的前驱和后继信息。这些信息只有在遍历的动态过程中才能得到,因此,引入线索二叉树来保存这些从动态过程中得到的信息。

2) 建立线索二叉树

为了保存结点在任一序列中的前驱和后继信息,可以考虑在每个结点中增加两个指针域来存放遍历时得到的前驱和后继信息,这样就可以为以后的访问带来方便。但增加指针信息会降低存储空间的利用率,因此可考虑采用其他方法。

若 n 个结点的二叉树采用二叉链表作存储结构,则链表中必然有 $n+1$ 个空指针域,可以充分利用这些空指针域来存放结点的前驱和后继信息。线索链表的结点结构如图 2-22 所示。

ltag	lchild	data	rchild	rtag
------	--------	------	--------	------

图 2-22 线索链表的结点结构

其中标志域:

$$\begin{aligned} \text{ltag} &= \begin{cases} 0 & \text{lchild 域指示结点的左孩子} \\ 1 & \text{lchild 域指示结点的直接前驱} \end{cases} \\ \text{rtag} &= \begin{cases} 0 & \text{rchild 域指示结点的右孩子} \\ 1 & \text{rchild 域指示结点的直接后继} \end{cases} \end{aligned}$$

若二叉树的二叉链表采用图 2-21 所示的结点结构,则相应的链表称为线索链表,其中指向结点前驱和后继的指针称为线索。加上线索的二叉树称为线索二叉树。对二叉树以某种次序遍历使其变为线索二叉树的过程称为线索化。中序线索二叉树及其存储结构如图 2-23 所示。

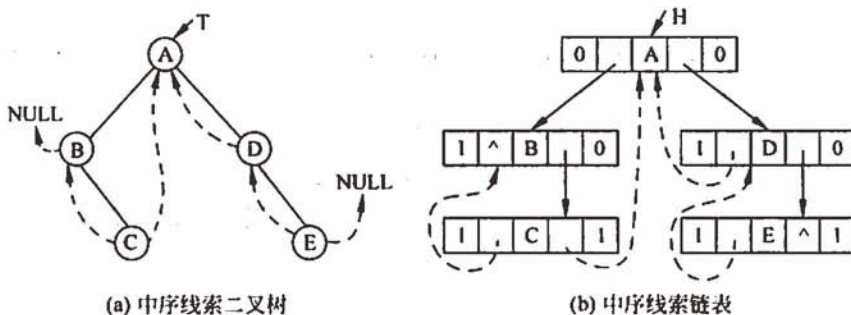


图 2-23 线索二叉树及其存储结构示意图

那么如何进行线索化呢? 因为按某种次序将二叉树线索化,实质上是在遍历过程中用线索取代空指针。因此,若设指针 p 指向正在访问的结点,则遍历时设立一个指针 pre ,使其始终指向刚刚访问过的结点,这样就记下了遍历过程中结点被访问的先后关系。

在遍历的过程中,设指针 p 指向正在访问的结点:

- (1) 若 p 指向的结点有空指针域,则将相应的标志域置为 1;
- (2) 若 $pre \neq \text{NULL}$ 且 pre 所指结点的 $rtag$ 等于 1,则 $pre \rightarrow rchild = p$;
- (3) 若 p 指向结点的 $ltag$ 等于 1,则令 $p \rightarrow lchild = pre$;
- (4) 使 pre 指向刚刚访问过的结点,即令 pre 等于 p 。

需要说明的是,用这种方法得到的线索二叉树,其线索是不完整的。也就是说,部分结点的前驱或后继信息还不能从其存储结构中直接得到。

3) 访问线索二叉树

如何在线索二叉树中查找结点的前驱和后继呢?

以中序线索二叉树为例,令 p 指向树中的某个结点。查找 p 所指结点的后继结点的方法是:

- (1) 若 $p \rightarrow rtag = 1$,则 $p \rightarrow rchild$ 即指向其后继结点;
- (2) 若 $p \rightarrow rtag = 0$,则 p 所指结点的中序后继必然是其右子树中进行中序遍历时得到的第一个结点。也就是说,从 p 所指结点的右子树的根结点出发,沿左孩子指针链向下查找,直至找到一个没有左孩子的结点时为止,这个结点就是 p 所指结点的直接后继结点,也称其为 p 的右子树中“最左下”的结点。

令 p 指向中序线索树中的某个结点, 则查找 p 所指结点的直接前驱的方法是:

(1) 若 $p \rightarrow \text{ltag} = 1$, 则 $p \rightarrow \text{lchild}$ 即指向其前驱结点;

(2) 若 $p \rightarrow \text{ltag} = 0$, 则 p 所指结点的中序前驱必然是其左子树中进行中序遍历时得到的最后一个结点。也就是说, 从 p 所指结点的左子树的根结点出发, 沿右孩子指针链向下查找, 直至找到一个没有右孩子的结点时为止, 这个结点就是 p 所指结点的直接前驱结点, 也称其为 p 的左子树中“最右下”的结点。

7. 二叉树的应用: 最优二叉树

1) 霍夫曼树

霍夫曼树又称最优二叉树, 是一种带权路径长度最短的树。

路径是从树中一个结点到另一个结点之间的通路, 路径上的分支数目称为路径长度。

树的路径长度是从树根到每一个叶子之间的路径长度之和。结点的带权路径长度为从该结点到树根之间的路径长度与该结点权的乘积。

树的带权路径长度为树中所有叶子结点的带权路径长度之和, 记为:

$$WPL = \sum_{k=1}^n w_k l_k$$

其中, n 为带权叶子结点数目, w_k 为叶子结点的权值, l_k 为叶子结点到根的路径长度。

霍夫曼树是指权值为 w_1, w_2, \dots, w_n 的 n 个叶子结点的二叉树中带权路径长度最小的二叉树。

例如, 在图 2-24 所示的具有 4 个叶子结点的二叉树中, 以图 2-24 (b) 所示的二叉树的带权路径长度最小。

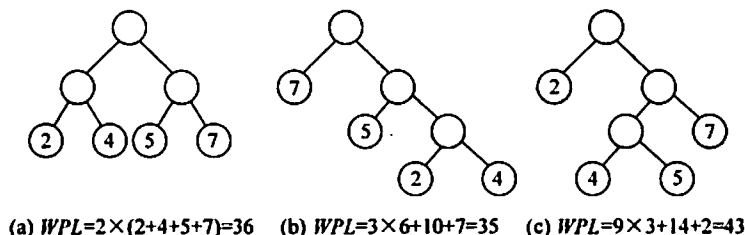


图 2-24 不同带权路径长度的二叉树

那么如何构造最优二叉树呢? 构造最优二叉树的霍夫曼算法如下:

① 根据给定的 n 个结点的权值 $\{w_1, w_2, \dots, w_n\}$ 构成 n 棵二叉树的集合 $F = \{T_1, T_2, \dots, T_n\}$, 其中每棵树 T_i 中只有一个权为 w_i 的根结点, 其左右子树均空。

② 在 F 中选取两棵根结点的权值最小的树作为左右子树, 并新生成一个根结点, 构造一棵新的二叉树, 置新构造二叉树的根结点的权值为其左、右子树根结点的权值之和。

③ 从 F 中删除这两棵树,同时将新得到的二叉树加入 F 中。

重复 ②、③,直到 F 中只含一棵树时为止。这棵树便是最优二叉树(霍夫曼树)。

由此算法可知,以选中的两棵子树构成新的二叉树,谁作为左子树,谁作为右子树,并没有明确。所以具有 n 个叶子结点的权值为 w_1, w_2, \dots, w_n 的最优二叉树(霍夫曼树)不惟一,但其 WPL 值是惟一确定的。

2) 霍夫曼编码

若对每个字符编制相同长度的二进制码,则称为等长编码。例如,英文字符集中的 26 个字符可用 5 位二进制串表示,按等长编码格式构造一个字符——编码表。发送方按照编码表对信息原文进行编码后送出电文,接收方对收到的二进制代码按每 5 位一组进行分割,通过查字符的编码表即可得到字符,实现译码。

等长编码方案的实现方法比较简单,但对通信中的原文进行编码后,所得电文的码串过长,不利于提高通信效率,因此希望缩短码串的总长度。如果对每个字符设计长度不等的编码,且让电文中出现次数较多的字符采用尽可能短的编码,则传送的电文总长度则可减少。

若要设计长度不等的编码,必须满足下面的条件:任一字符的编码都不是另一个字符的编码的前缀,这种编码也称为前缀码。对给定的字符集 $D = \{d_1, d_2, \dots, d_n\}$ 及字符的概率 $W = \{w_1, w_2, \dots, w_n\}$,构造其最优前缀码的方法为:以 d_1, d_2, \dots, d_n 作为叶子结点, w_1, w_2, \dots, w_n 作为叶子结点的权值,构造一棵最优二叉树。然后将树中每个结点的左分支标上“0”(或“1”),右分支标上“1”(或“0”),则每个叶子结点代表的字符的编码就是从根到叶子路径上的“0”、“1”字符组成的串。最优前缀编码方法如图 2-25 所示。

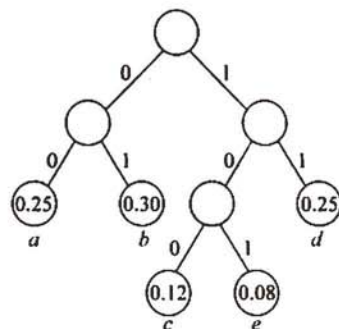


图 2-25 前缀编码示例

霍夫曼树也可以用来译码。译码过程为:从根结点出发,按二进制位串中的 0 和 1 确定进入左分支还是右分支,当到达叶子结点时译出该叶子对应的字符。若电文未完,则回到根结点继续进行上述过程。

8. 树和森林

1) 树的存储结构

(1) 树的双亲表示法:用一组地址连续的单元存储树的结点,并在每个结点中附设一个指示器,指示其双亲结点在该存储结构中的位置(结点所在数组元素的下标)。显然这种表示对于求指定结点的双亲或祖先都十分方便,但对于求指定结点的孩子及后代则

需要遍历整个数组,如图 2-26 所示。

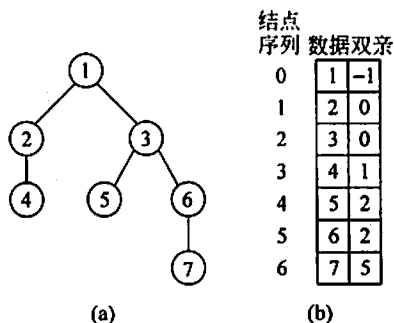


图 2-26 树的双亲表示法

(2) 树的孩子表示法: 在存储结构中用指针指出结点的每个孩子。由于树中每个结点的子树数目不尽相同,因此在采用链式存储结构时可以考虑多重链表。因为每个结点的指针数目不好确定,对于定长的结点,若依据树的度来设置结点中的指针,显然会造成极大的浪费。若设置的结点中的指针数目不等,则运算时又不方便。为此可以考虑为树中每个结点的孩子建立一个链表,即把每个结点的孩子结点看成一个线性表,则 n 个结点的树具有 n 个单链表。这 n 个单链表的头指针又可以排成一个线性表,如图 2-27(a) 所示。

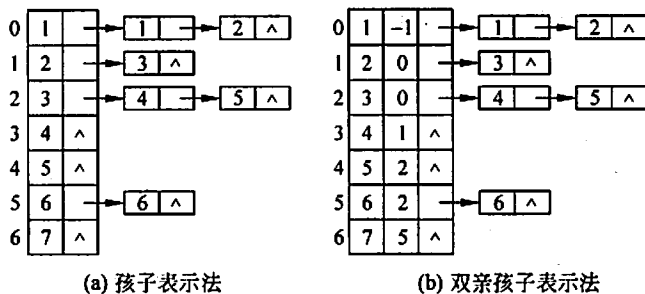


图 2-27 树的孩子表示法

也可以将双亲表示法和孩子表示法结合起来,形成树的双亲孩子表示结构,如图 2-27(b)所示。

(3) 孩子兄弟表示法: 又称二叉链表表示法。在链表的结点中设置两个指针域,分别指向该结点的第一个孩子和该节点的下一个兄弟。利用这种存储结构可实现各种树的操作。实际上,树的这种表示法为实现树、森林与二叉树之间的转换提供了基础。充分利用二叉树的有关算法来实现树及森林的操作,对难于把握规律的树和森林有着重要的现

实意义。树的孩子兄弟表示法如图 2-28 所示。

2) 树和森林的遍历

(1) 树的遍历。

由于树中每个结点可以有两棵以上的子树,因此遍历树的方法有两种:先根遍历和后根遍历。

- 树的先根遍历:先访问树的根结点,然后依次先根遍历根的各棵子树。对树的先根遍历等同于对转换所得的二叉树进行先序遍历。
- 树的后根遍历:先依次后根遍历树根的各棵子树,然后访问树根结点。树的后根遍历等同于对转换所得的二叉树进行中序遍历。

(2) 森林的遍历。

按照森林和树的相互递归的定义,可以得出森林的两种遍历方法。

- 先序遍历森林:若森林非空,访问森林中第一棵树的根结点,先序遍历第一棵树根结点的子树森林,再先序遍历除第一棵树之外的树所构成的森林。
- 中序遍历森林:若森林非空,中序遍历森林中第一棵树的子树森林,访问第一棵树的根结点,中序遍历除第一棵树之外的树所构成的森林。

3) 树、森林和二叉树之间的相互转换

树、森林和二叉树之间有一种自然的对应关系,它们之间可以互相进行转换。即任何一个森林或一棵树可以对应一棵二叉树,而任一棵二叉树也能对应到一个森林或一棵树上。

(1) 树、森林转换为二叉树。

利用树的孩子兄弟表示法可导出树与二叉树的对应关系。从物理结构上看,树的孩子兄弟表示法与二叉树的二叉链表表示法相同。因此,可以用这种同一存储结构的不同解释,将一棵树转换为一棵二叉树,如图 2-29 所示。一棵树可转换成惟一的一棵二叉树。

由于树根没有兄弟,所以树转换为二叉树后,二叉树的根一定没有右子树。因此,将

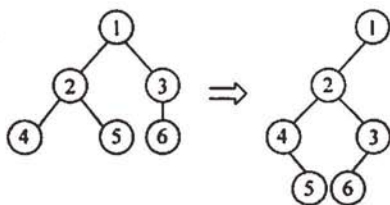


图 2-29 树转换为二叉树

一个森林转换为一棵二叉树的方法是:先将森林中的每一棵树转换为二叉树,再将第一棵树的根作为转换后的二叉树的根,第一棵树的左子树作为转换后二叉树根的左子树,第二棵树作为转换后二叉树根的右子树,第三棵树作为转换后二叉树根的右子树,依此类推,森林就可以转换为一棵二叉树,如图 2-30 所示。

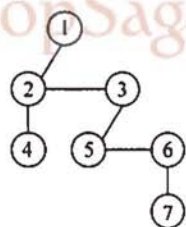


图 2-28 树的孩子兄弟表示法

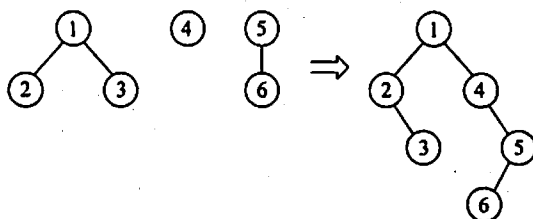


图 2-30 森林转换为二叉树

(2) 二叉树转换为树和森林。

若二叉树非空,则二叉树根及其左子树为第一棵树的二叉树形式,二叉树根的右子树又可以看作一个由森林转换后的二叉树。应用同样的方法,直到最后产生一棵没有右子树的二叉树为止,这样就得到了森林。为了进一步得到树,可用树的二叉链表表示的逆方法。即结点的右子树的根、右子树的右子树的根……原本是同一个双亲的兄弟,如图 2-31 所示。二叉树转换为树或森林是惟一的。

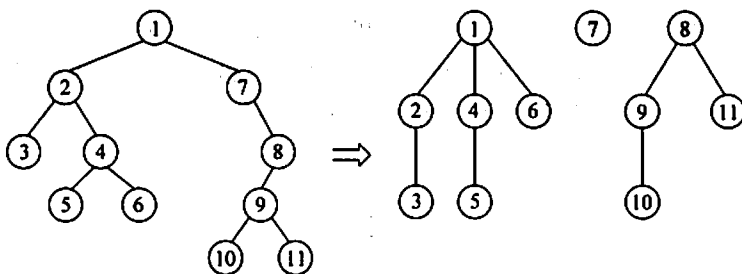


图 2-31 二叉树转换为树和森林

2.1.4 图

图是比树更复杂的一种数据结构。在线性结构中,除首结点没有前驱,末结点没有后继之外,一个结点只有惟一的一个直接前驱和惟一的一个直接后继。在树结构中,可认为除根结点没有前驱结点,其余每个结点只有惟一的一个前驱(双亲)结点和多个后继(子树)结点。而在图结构中,任意两个结点之间都可能直接有联系,所以图中一个结点的前驱结点和后继结点的个数是没有限制的。

1. 图的定义及基本运算

1) 图的定义及术语

图 G 是由两个集合 V 和 E 构成的二元组,记作 $G = (V, E)$,其中 V 是图中顶点的非空有限集合, E 是图中边的有限集合。从数据结构的逻辑关系角度来看,图中任一顶点都

可能与图中其他顶点有关系,而图中所有顶点都可能与某一顶点有关系。在图中,数据结构中的数据元素用顶点表示,数据元素之间的关系用边表示。

- 有向图: 若图中每条边都是有方向的,那么顶点之间的关系可用 $\langle v_i, v_j \rangle$ 表示,说明从 v_i 到 v_j 有一条有向边(也称为弧)。 v_i 是有向边的起点,称为弧尾; v_j 是有向边的终点,称为弧头。边有方向的图称为有向图。
- 无向图: 若图中的每条边都是无方向的,顶点 v_i 和 v_j 之间的边用 (v_i, v_j) 表示。因此,在有向图中的 $\langle v_i, v_j \rangle$ 与 $\langle v_j, v_i \rangle$ 分别表示两条边,而在无向图中的 (v_i, v_j) 与 (v_j, v_i) 表示的是同一条边。
- 无向完全图与有向完全图: 若一个无向图具有 n 个顶点,而每一个顶点与其他 $n-1$ 个顶点之间都有边,则称之为无向完全图。显然,含有 n 个顶点的无向完全图共有 $\frac{n(n-1)}{2}$ 条边。类似地,在有 n 个顶点的有向完全图中,弧的数目为 $n(n-1)$,即任两个不同顶点之间都有方向相反的两条弧存在。
- 度、出度和入度: 顶点 v 的度是指关联于该顶点的边的数目,记作 $D(v)$ 。若 G 为有向图,顶点的度表示该顶点的入度和出度之和。顶点的入度是以该顶点为终点的有向边的数目,而顶点的出度指以该顶点为起点的有向边的数目,分别记为 $ID(v)$ 和 $OD(v)$ 。无论是有向图还是无向图,顶点数 n 、边数 e 与各顶点的度之间有以下关系:

$$e = \frac{1}{2} \sum_{i=1}^n D(v_i)$$

- 路径: 在无向图 G 中,从顶点 v_p 到顶点 v_q 的路径是指存在一个顶点序列 $v_p, v_{i_1}, v_{i_2}, \dots, v_{i_n}, v_q$,使得 $(v_p, v_{i_1}), (v_{i_1}, v_{i_2}), \dots, (v_{i_n}, v_q)$ 均属于 $E(G)$ 。若 G 是有向图,其路径也是有方向的,它由 $E(G)$ 中的有向边 $\langle v_p, v_{i_1} \rangle, \langle v_{i_1}, v_{i_2} \rangle, \dots, \langle v_{i_n}, v_q \rangle$ 组成。路径长度是路径上的边或弧的数目。第一个顶点和最后一个顶点相同的路径称为回路或环。若一条路径上除了 v_p 和 v_q 相同外,其余顶点均不相同,这种路径称为简单路径。
- 子图: 若有两个图 $G=(V, E)$ 和 $G'=(V', E')$,如果 $V' \subseteq V$ 且 $E' \subseteq E$,则称 G' 为 G 的子图。
- 连通图与连通分量: 在无向图 G 中,若从顶点 v_i 到顶点 v_j 有路径,则称顶点 v_i 和顶点 v_j 是连通的。如果无向图 G 中任意两个顶点都是连通的,则称其为连通图。无向图 G 的极大连通子图称为 G 的连通分量。
- 强连通图与强连通分量: 在有向图 G 中,如果对于每一对顶点 $v_i, v_j \in V$ 且 $v_i \neq v_j$,从顶点 v_i 到顶点 v_j 和从顶点 v_j 到顶点 v_i 都存在路径,则称图 G 为强连通图。

有向图中的极大连通子图称为有向图的强连通分量。

- 网：边（或弧）带权值的图称为网。
- 生成树：一个连通图的生成树是一个极小的连通子图，它包含图中的全部顶点，但只有构成一棵树的 $n-1$ 条边。
- 有向树和生成森林：如果一个有向图恰有一个顶点的入度为 0，其余顶点的入度均为 1，则是一棵有向树。有向图的生成森林由若干棵有向树组成，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧。

从图的逻辑结构的定义来看，图中的顶点之间不存在全序关系（即无法将图中的顶点排列成一个线性序列），任何一个顶点都可被看成第一个顶点。另一方面，任一顶点的邻接点之间也不存在次序关系。为了便于运算，我们给图中每个顶点赋予一个序号值。

2) 图的基本运算

图的运算有许多，大致可分为 4 类。

- (1) 建图与置空。
- (2) 查询运算：查询有关顶点的信息及其邻接点等。
- (3) 插入和删除：插入及删除图中的顶点和边（或弧）。
- (4) 遍历图：按照某种策略访问图中的全部顶点且仅访问 1 次。有两种基本的遍历方法：深度优先搜索和广度优先搜索。

2. 图的存储结构

1) 邻接矩阵表示法

邻接矩阵用于表示图中顶点之间的关系。对于具有 n 个顶点的图 $G=(V, E)$ 来说，其邻接矩阵是一个 n 阶方阵，且满足：

$$A[i][j] = \begin{cases} 1 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E \text{ 中的边} \\ 0 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E \text{ 中的边} \end{cases}$$

由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵就不一定对称了。借助于邻接矩阵，很容易判定任意两个顶点之间是否有边（或弧）相连，并且容易求得各个顶点的度。对于无向图，顶点 v_i 的度是邻接矩阵中第 i 行（或列）的值不为 0 的元素个数，即元素之和。对于有向图，第 i 行的元素之和是顶点 v_i 出度 $OD(v_i)$ ，第 j 列的元素之和为顶点 v_j 的入度 $ID(v_j)$ 。图 2-32 所示的有向图和无向图的邻接矩阵分别为 A 和 B 。

网（赋权图）的邻接矩阵可定义为：

$$A[i][j] = \begin{cases} W_{ij} & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E \text{ 中的边} \\ \infty & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E \text{ 中的边} \end{cases}$$

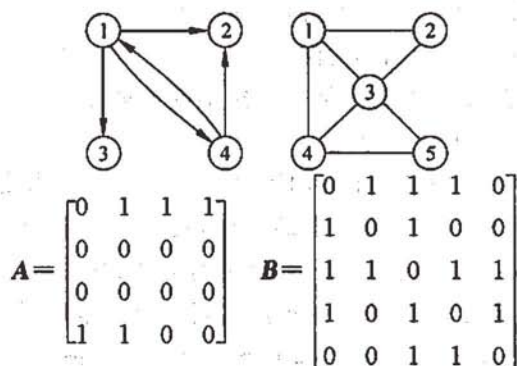


图 2-32 有向图和无向图

图 2-33 所示的是网及其邻接矩阵 C 。

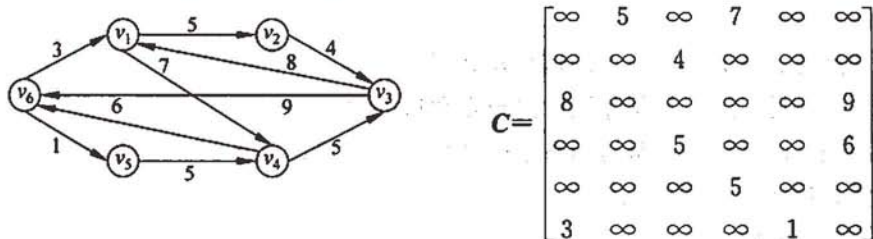


图 2-33 一个网及其邻接矩阵表示

另外还有路径邻接矩阵。设 G 是具有 n 个顶点 $\{v_1, v_2, \dots, v_n\}$ 的图, 其邻接矩阵为 A , 则 $A^L (L=1, 2, \dots)$ 中的 (i, j) 项元素 $a_{ij}^{(L)}$ 是从顶点 v_i 到顶点 v_j 的长度为 L 的路径的总数。

2) 邻接链表表示法

邻接链表指的是: 为图的每个顶点建立一个单链表, 第 i 个单链表中的结点表示依附于顶点 v_i 的边 (对于有向图是以 v_i 为尾的弧)。邻接链表中的结点有表结点和表头结点两种类型, 如下所示。

表结点		
adjvex	nextarc	info

表头结点	
data	firstarc

其含义为:

adjvex 指示与顶点 v_i 邻接的顶点的序号;

nextarc 指示下一条边或弧的结点;

info 存储和边或弧有关的信息, 如权值等;

data 存储顶点 v_i 的名或其他有关信息;

firstarc 指示链表中的第一条边或弧。

这些表头结点(可以链相连)通常以顺序结构的形式存储,以便随机访问任一顶点的链表。显然,对于有 n 个顶点、 e 条边的无向图来说,其邻接链表需用 n 个头结点和 $2e$ 个表结点。

对于无向图的邻接链表,顶点 v_i 的度恰为第 i 个邻接链表中表结点的数目。而在有向图中,为求顶点的入度,必须扫描每个邻接表,这是因为第 i 个邻接链表中表结点的数目只是顶点 v_i 的出度。为此,可以建立一个有向图的逆邻接链表。有向图的邻接表和逆邻接表如图 2-34 所示。

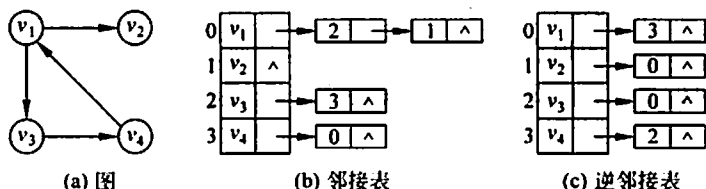


图 2-34 一个图及其邻接表和逆邻接表

以上两种表示法对有向图和无向图都适用。

3. 图的遍历

与树的遍历相似,图的遍历也是从某个顶点出发,沿着某条搜索路径对图中所有顶点进行访问且只访问一次。图的遍历算法是求解图的连通性问题、拓扑排序及求关键路径等算法的基础。

图的遍历要比树的遍历复杂得多。因为图的任一个结点都可能与其余顶点相邻接,所以在访问了某个顶点之后,可能会沿着某路径又回到该结点上。为了避免顶点的重复访问,在图的遍历过程中,必须记下每个已访问过的顶点。

1) 深度优先搜索(DFS)

此种方法类似于树的先根遍历,也是第一次经过一个顶点时就进行访问操作。从图 G 中任一结点 v 出发按深度优先搜索法进行遍历的步骤如下:

① 设立搜索指针 p ,使 p 指向顶点 v ;

② 访问 p 结点,并使 p 指向与 p 顶点相邻接的且尚未被访问过的结点;

③ 若 p 不空,则重复步骤②,否则执行步骤④;

④ 沿着刚才访问的次序和方向回溯到一个尚有邻接顶点且未被访问过的顶点,并使 p 指向这个未被访问的邻接顶点,然后重复步骤②,直至所有的顶点均被访问为止。

该算法的特点是尽可能先对纵深方向搜索,因此可以很容易地得到递归的遍历算法。

深度优先遍历图的过程实质上是对某个顶点查找其邻接点的过程,其耗费的时间取决于所采用的存储结构。当图用邻接矩阵表示时,查找所有顶点的邻接点所需时间为 $O(n^2)$ 。若以邻接表作为图的存储结构,则需要 $O(e)$ 的时间复杂度查找所有顶点的邻接点。因此,当以邻接表作为存储结构时,深度优先搜索遍历图的时间复杂度为 $O(n+e)$ 。深度优先的搜索过程如图 2-35 所示。

2) 广度优先搜索(BFS)

此种方法类似于树的按层次遍历的过程。其过程如下:

假设从图中某个顶点 v 出发,在访问了 v 之后依次访问 v 的各个未被访问过的邻接点。然后分别从这些邻接点出发依次访问它们的邻接点,并使“先访问的顶点的邻接点”先于“后访问的顶点的邻接点”被访问,直至图中所有已被访问的顶点的邻接点都被访问到。若此时还有未访问的顶点,则另选其中一个作为起点,重复上述过程,直至图中所有的顶点都被访问到为止。图 2-36 所示的图的广度优先搜索序列为: 1, 2, 3, 4, 5, 6。

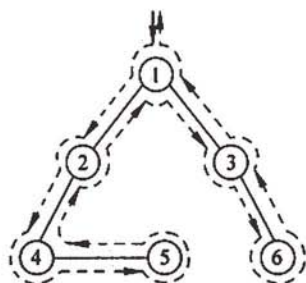


图 2-35 深度优先搜索遍历过程

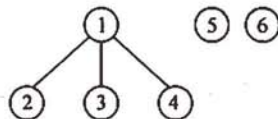


图 2-36 一个不连通的无向图

广度优先遍历图的特点是尽可能先进行横向搜索,即最先访问的顶点的邻接点亦先被访问。为此,引入队列结构来保存已访问过的顶点序列。即访问一个顶点后,就将其放入队中。当队头顶点出队时,就访问其未被访问的邻接点并令这些邻接点入队。

在广度优先遍历算法中,每个顶点至多进一次队列。遍历图的过程实质上是通过边或弧找邻接点的过程。因此广度优先搜索遍历图和深度优先搜索遍历图的时间复杂度相同,其不同之处仅仅在于对顶点访问的顺序不同。

4. 生成树及最小生成树

1) 生成树的概念

设图 $G=(V, E)$ 是个连通图,当从图中任一顶点出发遍历图 G 时,将边集 $E(G)$ 分为两个集合 $A(G)$ 和 $B(G)$ 。其中 $A(G)$ 是遍历时所经过的边的集合, $B(G)$ 是遍历时未经过的边的集合。显然, $G_1=(V, A)$ 是图 G 的子图,称子图 G_1 为连通图 G 的生成树。图 2-37 所示的是图及其生成树和非生成树。

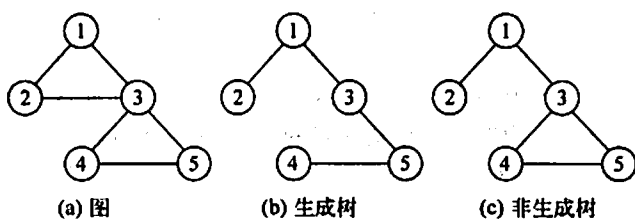


图 2-37 一个无向图的生成树和非生成树

按深度和广度优先搜索分别进行遍历将得到不同的生成树,分别称为深度优先生成树和广度优先生成树。例如,图 2-38 所示的是图 2-37(a)对应的一棵深度优先生成树和一棵广度优先生成树。

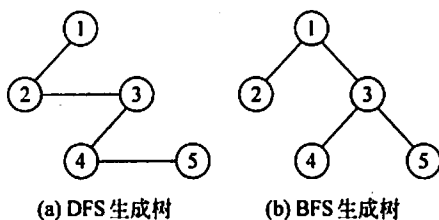


图 2-38 图的搜索生成树

有 n 个顶点的连通图至少有 $n-1$ 条边,而生成树中恰好有 $n-1$ 条边,所以连通图的生成树是该图的极小连通子图。若在图的生成树中任意加一条边,则必然形成回路。

图的生成树不是惟一的。从不同的顶点出发,选择不同的存储方式,可以得到不同的生成树。对于非连通图而言,每个连通分量中的顶点集和遍历时走过的边集一起构成若干棵生成树,称为非连通图的生成树森林。

2) 最小生成树

由于生成树不惟一,从不同的顶点出发可得到不同的生成树。对于连通网来说,边是带权值的,生成树的各边也带权值,于是就把生成树各边的权值总和称为生成树的权,把权值最小的生成树称为最小生成树。求解最小生成树有许多实际的应用。

构造最小生成树有多种算法,其中多数算法利用了最小生成树的 MST 性质:假设 $N=(V,E)$ 是一个连通图, U 是顶点集 V 的一个非空子集,若 (u,v) 是一条最小权值的边,其中 $u \in U, v \in V-U$,则必存在一棵包含边 (u,v) 的最小生成树。此性质使用反证法很容易得到证明。

(1) 普里姆(Prim)算法。

假设 $N=(V,E)$ 是连通网, TE 是 N 上最小生成树中的边的集合。算法从顶点集合

$U = \{u_0\} (u_0 \in V)$ 、边的集合 $TE = \{\}$ 开始, 重复执行下述操作: 在所有 $u \in U, v \in V - U$ 的边 $(u, v) \in E$ 中找一条代价最小的边 (u_0, v_0) , 把这条边并入集合 TE , 同时将 v_0 并入集合 U , 直至 $U = V$ 时为止。此时 TE 中必有 $n-1$ 条边, 则 $T = (V, \{TE\})$ 为 N 的最小生成树。

由此可知, 普里姆算法构造最小生成树的过程是以一个顶点集合 $U = \{u_0\}$ 作初态, 不断寻找与 U 中顶点相邻且代价最小的边的另一个顶点, 扩充 U 集合直至 $U = V$ 时为止。

用普里姆算法构造最小生成树的过程如图 2-39 所示。

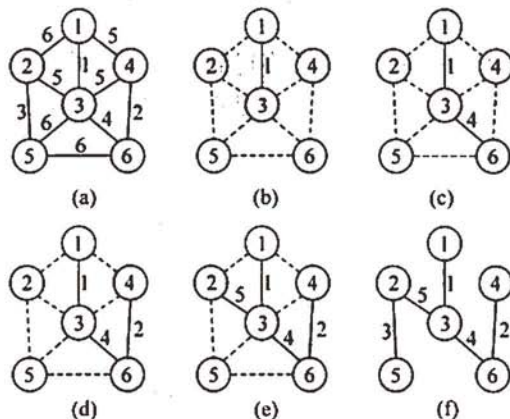


图 2-39 普里姆算法构造最小生成树的过程

普里姆算法的时间复杂度为 $O(n^2)$, 与图中的边数无关, 因此该算法适合于求边稠密的网的最小生成树。

(2) 克鲁斯卡尔 (Kruskal) 算法。

克鲁斯卡尔求最小生成树的算法思想为: 假设连通网 $N = (V, E)$, 令最小生成树的初始状态为只有 n 个顶点而无边的非连通图 $T = (V, \{\})$, 图中每个顶点自成一个连通分量。在 E 中选择代价最小的边, 若该边依附的顶点落在 T 中不同的连通分量上, 则将此边加入 T 中, 否则舍去此边而选择下一条代价最小的边。依次类推, 直至 T 中所有顶点都在同一连通分量上为止。

用克鲁斯卡尔算法构造最小生成树的过程如图 2-40 所示。

克鲁斯卡尔算法的时间复杂度为 $O(e \log e)$, 与图中的顶点数无关, 因此该算法适合于求边稀疏的网的最小生成树。

5. 拓扑排序和关键路径

1) AOV 网

在工程领域, 一个大的工程项目通常被划分为许多较小的子工程 (称为活动)。显然, 当这些子工程都完成时, 整个工程也就完成了。在有向图中, 若以顶点表示活动, 用有

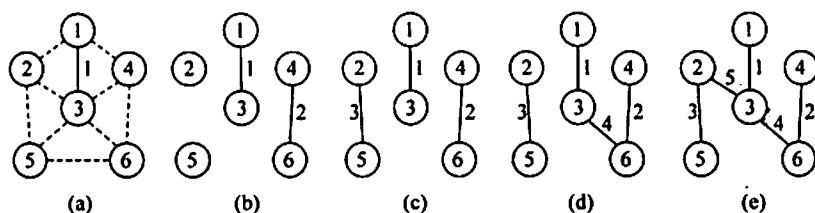


图 2-40 克鲁斯卡尔算法构造最小生成树的过程

向边表示活动之间的优先关系,则称这样的有向图为以顶点表示活动的网(Activity On Vertex network),简称 AOV 网。在网中,若从顶点 v_i 到顶点 v_j 有一条有向路径,则顶点 v_i 是 v_j 的前驱,顶点 v_j 是 v_i 的后继。若 $\langle v_i, v_j \rangle$ 是网中的一条弧,则顶点 v_i 是 v_j 的直接前驱,顶点 v_j 是 v_i 的直接后继。AOV 网中的弧表示了活动之间的优先关系,也可以说是一种活动进行时的制约关系。

AOV 网中不应出现有向环,若存在的话,则意味着某项活动必须以自身任务的完成为先决条件,显然这是荒谬的。因此,若要检测一个工程是否可行,首先就应检查对应的 AOV 网是否存在回路。不存在回路的 AOV 网称为有向无环图,或 DAG (directed acycline graph) 图。检测的方法是对有向图构造其顶点的拓扑有序序列。若图中所有顶点都在它的拓扑有序序列中,则该 AOV 网中必定不存在环。

2) 拓扑排序及其算法

拓扑排序是将 AOV 网中所有顶点排成一个线性序列,该序列满足:若在 AOV 网中从顶点 v_i 到 v_j 有一条路径,则在该线性序列中,顶点 v_i 必然在顶点 v_j 之前。

拓扑排序即指对 AOV 网构造拓扑序列的操作。一般情况下,假设 AOV 图代表一个工程计划,则 AOV 网的一个拓扑排序就是一个工程顺利完成的可行方案。

对 AOV 网进行拓扑排序的方法如下:

- ① 在 AOV 网中选择一个入度为零(没有前驱)的顶点且输出它;
- ② 从网中删除该顶点及与该顶点有关的所有边;
- ③ 重复上述两步,直至网中不存在入度为零的顶点为止。

执行的结果会有两种情况:一种是所有顶点已输出,此时整个拓扑排序完成,说明网中不存在回路;另一种是尚有未输出的顶点,剩余的顶点均有前驱顶点,表明网中存在回路,拓扑排序无法进行下去。图 2-41 所示的是拓扑排序过程,得到的拓扑序列为: 6, 1, 4, 3, 2, 5。

若在 AOV 网中考察各顶点的出度,并以下列步骤进行排序,则这种排序称为逆拓扑:

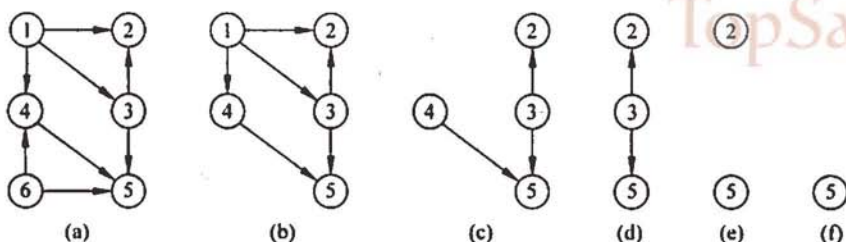


图 2-41 拓扑排序过程

- ① 在 AOV 网中,选一个没有后继的顶点(出度为零)且输出它;
- ② 从网中删除该顶点,并删去所有到达该顶点的弧(即修改该顶点的直接前驱顶点的出度);

③ 重复上述两步,直至网中没有出度为零的顶点为止。

若这时全部顶点均被输出,则排序成功。若不能输出全部顶点,则说明网中存在有向回路。

当有向图中无环时,也可以利用深度优先遍历进行逆拓扑排序。因为图中无环,则由图中某点出发进行深度优先搜索遍历时,最先退出 DFS 函数的顶点即是出度为零的顶点,它是拓扑有序序列中最后的一个顶点。由此,按退出 DFS 函数的先后记录下来的顶点序列即为逆向的拓扑有序序列。

拓扑排序算法的时间复杂度为 $O(n+e)$ 。

3) AOE 网

若在带权有向图 G 中以顶点表示事件,以有向边表示活动,边上的权值表示该活动持续的时间,则这种带权有向图称为用边表示活动的网(activity on edge network),简称 AOE 网。通常在 AOE 网上列出完成预定工程计划所需要进行的各项活动,每项活动的计划完成时间,要发生哪些事件,以及这些事件和活动间的关系,从而可以分析该项工程是否实际可行,估计工程完成的最短时间以及哪些活动是影响工程进度的关键。进一步可以进行人力、物力的调度和分配,以达到缩短工期的目的。

在用 AOE 网表示一项工程计划时,顶点表示的事件实际上就是某些活动已经完成,以及某些子工程可以动工的标志。具体地说,顶点表示的事件是指该顶点所有进入边所表示的活动均已完成,以及它的发出边所表示的活动均可以开始的一种状态。

对于一个工程来说,一般有一个开始状态和一个结束状态,所以在 AOE 网中至少有一个开始顶点。开始顶点的入度为零,亦称为源点。另外,还应有一个结束顶点。结束顶点的出度为零,亦称为汇点。AOE 网中不应存在有向回路,否则整个工程无法完成。

与 AOV 网不同,AOE 网所关心的是:

(1) 完成该工程至少需要多少时间?

(2) 哪些活动是影响整个工程进度的关键?

由于 AOE 网中的某些活动能够并行地进行,所以完成整个工程所需的时间是从开始顶点到结束顶点的最长路径的长度。这里的路径长度是指该路径上的权值(时间)之和。

4) 关键路径和关键活动

从源点到汇点的路径长度最长的路径称为关键路径。关键路径上的所有活动均是关键活动。如果任何一项关键活动没有按期完成,则会影响到整个工程的进度。而提高关键活动的速度通常可以缩短整个工程的工期。

(1) 顶点事件的最早发生时间 $ve(j)$ 。 $ve(j)$ 是指从源点 v_0 到 v_j 的最长路径长度(时间)。这个时间决定了所有从 v_j 发出的弧所表示的活动能够开工的最早时间。

$ve(j)$ 的计算方法为:

$$\begin{cases} ve(0) = 0 \\ ve(j) = \max\{ve(i) + dut(<i, j>)\} & <i, j> \in T, 1 \leq j \leq n-1 \end{cases}$$

其中, T 是所有到达顶点 j 的弧的集合, $dut(<i, j>)$ 是弧 $<i, j>$ 上的权值, n 是网中的顶点数(即汇点的序号),如图 2-42 所示。

显然,上式是一个从源点开始的递推公式。 $ve(j)$ 的计算必须在 v_j 的所有前驱顶点的最早发生时间全部求出后才能进行。这样必须对 AOE 网进行拓扑排序,然后按拓扑有序序列逐个求出各顶点事件的最早发生时间。

(2) 顶点事件的最晚发生时间 $vl(i)$ 。 $vl(i)$ 是指在不推迟整个工程完成日期的前提下,事件 v_i 所允许的最晚发生时间。对一个工程来说,计划用几天时间完成可以从 AOE 网求得,其数值就是汇点 v_{n-1} 的最早发生时间 $ve(n-1)$,而这个时间也就是 $vl(n-1)$ 。其他顶点事件的 vl 应从汇点开始,逐步向源点方向递推才能求得,所以 vl 的计算公式为:

$$\begin{cases} vl(n-1) = ve(n-1) \\ vl(i) = \min\{vl(j) - dut(<i, j>)\} & <i, j> \in S, 0 \leq i \leq n-2 \end{cases}$$

其中, S 是所有从顶点 i 发出的弧的集合,如图 2-43 所示。

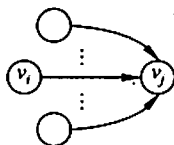


图 2-42 顺推事件的最早发生时间

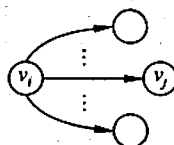


图 2-43 逆推事件的最晚发生时间

显然,上式的计算必须在顶点 v_i 的所有后继顶点的最晚发生时间全部求出后才能进行。这样必须对 AOE 网进行逆拓扑排序,由逆拓扑序列递推计算各顶点的 vl 值。

(3) 活动 a_k 的最早开始时间 $e(k)$ 。 $e(k)$ 是指弧 $\langle i, j \rangle$ 所表示的活动 a_k 最早可开工时间。

$$e(k) = ve(i)$$

这说明活动 a_k 的最早开始时间等于事件 v_i 的最早发生时间。这与 AOE 网中关于事件的含义的解释是完全一致的。

(4) 活动 a_k 的最晚开始时间 $l(k)$ 。 $l(k)$ 是指在不推迟整个工程完成日期的前提下,允许该活动最晚开始的时间。若活动 a_k 由弧 $\langle i, j \rangle$ 表示,则

$$l(k) = vl(j) - dut(\langle i, j \rangle)$$

对于活动 a_k 来说,若 $e(k) = l(k)$,则表示活动 a_k 是关键活动,它说明该活动最早可开工时间与整个工程计划允许该活动最晚的开工时间一致,施工期一点也不能拖延。若活动 a_k 不能按期完成,则工程将延期。若活动 a_k 提前完成,则可能使整个工程提前完工。

由关键活动组成的路径是关键路径。依照上述计算关键活动的方法,就可形成求 AOE 网的关键路径。

6. 最短路径

1) 单源点最短路径

所谓单源点最短路径是指:给定带权有向图 G 和源点 v ,求从 v 到 G 中其余各顶点的最短路径。迪杰斯特拉(Dijkstra)提出了按路径长度递增的次序产生最短路径的算法。设源点为 v_0 ,网中的权值均不小于 0,迪杰斯特拉算法的思想是:把网中所有的顶点分成两个集合 S 和 T , S 集合的初态只包含顶点 v_0 , T 集合的初态为网中除 v_0 之外的所有顶点的集合。凡以 v_0 为源点,已经确定了最短路径的终点并入 S 集合中。顶点集合 T 则是尚未确定最短路径的顶点的集合。按各顶点与 v_0 间最短路径长度递增的次序,逐个把 T 集合中的顶点加到 S 集合中去,使得从 v_0 到 S 集合中各顶点的路径长度始终不大于从 v_0 到 T 集合中各顶点的路径长度。为了能方便地求出从 v_0 到 T 集合中各顶点最短路径的递增次序,算法引入了一个辅助向量 $dist$ 。它的某一个分量 $dist[i]$ 表示当前求出的从 v_0 到终点 v_i 的最短路径长度。这个路径长度并不一定是真正最短路径长度。它的初始状态为:若从 v_0 到 u 有弧,则 $dist[u]$ 为弧上的权值。否则,置 $dist[i]$ 为 ∞ 。显然,长度为:

$$dist[u] = \min\{dist[i] \mid v_i \in V(G)\}$$

的路径就是从 v_0 出发的长度最短的一条最短路径。此路径为 (v_0, u) 。这时顶点 u 应从集合 T 中删除,将其并入集合 S 。

设网用邻接矩阵 $arcs$ 存储,那么每次选出一个顶点 u 并使之并入集合 S 后,就修改 T 集合中各顶点的最短路径长度 $dist$ 。对于 T 集合中的某一个顶点 i 来说,其最短路径

或者是 (v_0, v_i) ,或者是 (v_0, v_u, v_i) ,绝对不可能有其他选择。也就是说,若:

$$dist[u] + arcs[u][i] < dist[i]$$

则修改 $dist[i]$,使:

$$dist[i] = dist[u] + arcs[u][i]$$

对 T 集合中各顶点的 $dist$ 进行修改后,再从中挑选出一个路径长度最小的顶点,从 T 集合中删除之,然后将其并入 S 集合。依次类推,就能求出源点到其余各顶点的最短路径长度。

2) 每对顶点间的最短路径

若每次以一个顶点为源点,重复执行迪杰斯特拉算法 n 次,便可求得网中每一对顶点之间的最短路径。这里介绍弗洛伊德(Floyd)提出的求最短路径的算法,该算法在形式上要简单一些。

弗洛伊德算法的思想是:假设需要求从顶点 v_i 到 v_j 的最短路径。图的边信息采用邻接矩阵的方式存储, $arcs[i][j]$ 表示弧 (v_i, v_j) 上的权值。若此弧不存在,则权值为有别于有效权值的一个数。如果有 v_i 到 v_j 的弧,则从 v_i 到 v_j 存在一条长度为 $arcs[i][j]$ 的路径。该路径不一定是最短路径,尚需进行 n 次试探。首先考虑路径 (v_i, v_0, v_j) 是否存在(即判别弧 (v_i, v_0) 和 (v_0, v_j) 是否存在)。如果存在,则比较 (v_i, v_j) 和 (v_i, v_0, v_j) 的路径长度,取较短者为从 v_i 到 v_j 的中间顶点的序号不大于 0 的最短路径。假如在路径上再增加一个顶点 v_1 ,也就是说,如果 (v_i, \dots, v_1) 和 (v_1, \dots, v_j) 分别是当前找到的中间顶点的序号不大于 0 的最短路径,那么 $(v_i, \dots, v_1, \dots, v_j)$ 就有可能是从 v_i 到 v_j 的中间顶点的序号不大于 1 的最短路径。将它和已经得到的从 v_i 到 v_j 的中间顶点的序号不大于 0 的最短路径相比较,从中选出中间顶点的序号不大于 1 的最短路径之后,再增加一个顶点 v_2 继续进行试探。依次类推。在一般情况下,若 (v_i, \dots, v_k) 和 (v_k, \dots, v_j) 分别是从小 v_i 到 v_k 和从 v_k 到 v_j 的中间顶点的序号不大于 $k-1$ 的最短路径,则将 $(v_i, \dots, v_k, \dots, v_j)$ 与已经得到的从 v_i 到 v_j 的中间顶点的序号不大于 $k-1$ 的最短路径相比较,其长度较短者便是从 v_i 到 v_j 的中间顶点的序号不大于 k 的最短路径。这样,在经过 n 次比较后,最后求得的必是从 v_i 到 v_j 的最短路径。按此方法,可以同时求得各对顶点间的最短路径。

2.1.5 查找

1. 查找的基本概念及性能分析

1) 基本概念

查找是一种常用的基本运算,在程序设计中,特别是在非数值处理中,查找是最重要的一种运算。

查找表是指由同一类型的数据元素(或记录)构成的集合。“集合”这种数据结构中的

数据元素之间存在着完全松散的关系,因此,查找表是一种非常灵活的数据结构。

(1) 静态查找表:对查找表经常要进行两种操作,包括①查询某个“特定”的数据元素是否在查找表中;②检索某个“特定”的数据元素的各种属性。通常将只进行这两种操作的查找表称为静态查找表。

(2) 动态查找表:对查找表经常还要进行另外两种操作,包括③在查找表中插入一个数据元素;④从查找表中删除一个数据元素。若在查找过程中同时插入查找表中不存在的数据元素,或者从查找表中删除已存在的某个数据元素,则称此类查找为动态查找。

(3) 关键字:是数据元素(或记录)的某个数据项的值,用它来识别(标识)这个数据元素(或记录)。

(4) 主关键字:指能惟一标识一个数据元素的关键字。

(5) 次关键字:指能标识多个数据元素的关键字。

(6) 查找:根据给定的某个值,在查找表中确定是否存在一个其关键字等于给定值的记录或数据元素。若表中存在这样的一个记录,则称查找成功。此时或者给出整个记录的信息,或者指出记录在查找表中的位置。若表中不存在关键字等于给定值的记录,则称查找不成功。此时的查找结果用一个“空”记录或“空”指针表示。

2) 查找操作的性能分析

对于查找算法来说,其基本操作是“将记录的关键字与给定值进行比较”。因此,通常以“其关键字和给定值进行过比较的记录个数的平均值”作为衡量查找算法好坏的依据。

平均查找长度:为确定记录在查找表中的位置,须与给定关键字值进行比较的次数的期望值称为查找算法在查找成功时的平均查找长度。

对于含有 n 个记录的表,查找成功时的平均查找长度定义为:

$$ASL = \sum_{i=1}^n P_i C_i$$

其中: P_i 为对表中第 i 个记录进行查找的概率,且 $\sum_{i=1}^n P_i = 1$ 。一般情况下,均认为查找每个记录的概率是相等的,即 $P_i = 1/n$; C_i 为找到表中其关键字与给定值相等的记录(为第 i 个记录)时,和给定值已进行过比较的关键字的个数。显然, C_i 随查找方法的不同而不同。

2. 静态查找表

1) 顺序查找

(1) 顺序查找的基本思想。

从表中的一端开始,逐个对记录的关键字和给定值进行比较。若找到一个记录的关键字与给定值相等,则查找成功。若整个表中的记录均比较过,仍未找到关键字等于给定值的记录,则查找失败。

顺序查找的方法对于顺序存储方式和链式存储方式的查找表都适用。

(2) 顺序查找的性能分析。

从顺序查找的过程可见, C_i 取决于所查记录在表中的位置。若需要查找的记录正好是表中的第一个记录时, 仅需比较一次。若查找成功时找到的是表中的最后一个记录, 则需比较 n 次。一般情况下, $C_i = n - i + 1$, 因此在等概率情况下, 顺序查找成功的平均查找长度为:

$$ASL_s = \sum_{i=1}^n P_i C_i = \frac{1}{n} \sum_{i=1}^n (n - i + 1) = \frac{n+1}{2}$$

也就是说, 成功查找的平均比较次数约为表长的一半。若所查记录不在表中, 则必须进行 $n+1$ 次比较才能确定失败。

与其他查找方法相比, 顺序查找方法在 n 值较大时, 其平均查找长度较大, 查找效率较低。但这种方法也有优点, 那就是算法简单且适应面广, 对查找表的结构没有要求, 无论记录是否按关键字有序排列均可应用。

2) 折半查找

(1) 折半查找的基本思想。

设查找表的元素存储在一维数组 $r[1 \cdots n]$ 中, 如果在表中的元素已经按关键字递增(或递减)的方式排序的情况下, 就可以进行折半查找。其方法是: 首先将待查的关键字 key 值与表 r 中间位置上(下标为 mid)的记录的关键字进行比较。若相等, 则查找成功。若 $key > r[mid].key$, 则说明待查记录只可能在后半个子表 $r[mid+1] \cdots r[n]$ 中, 下一步应在后半个子表中再进行折半查找。若 $key < r[mid].key$, 说明待查记录只可能在前半个子表 $r[1] \cdots r[mid-1]$ 中, 下一步应在 r 的前半个子表中进行折半查找。这样通过逐步缩小范围, 直到查找成功或子表为空时失败为止。

可见折半查找过程是以处于中间位置记录的关键字和给定值进行比较。若相等, 则查找成功; 若不等, 则缩小范围, 直至新的区间中间位置记录的关键字等于给定值或者查找区间的大小为零时(表明查找不成功)为止。

(2) 折半查找的性能分析。

折半查找的过程可以用一棵二叉树描述, 方法是: 以当前查找区间的中间位置上的记录作为根, 左子表和右子表中的记录分别作为根的左子树和右子树上的结点, 这样构造的二叉树称为折半查找判定树。例如, 具有 11 个结点的折半查找判定树如图 2-44 所示。

从折半查找判定树可以看出, 查找成功时, 折半查找的过程恰好走了一条从根结点到被查结点的路径, 关键字比较的次数即为被查结点在树中的层数。因此, 折半查找在查找成功时进行比较的关键字数最多不超过树的深度。而具有 n 个结点的判定树的深度为 $\lfloor \log_2 n \rfloor + 1$, 所以折半查找在查找成功时和给定值进行比较的关键字数至

多为 $\lfloor \log_2 n \rfloor + 1$ 。

给判定树中所有结点的空指针域加上一个指向一个方形结点的指针,称这些方形结点为判定树的外部结点(与之相对应,称那些圆形结点为内部结点),如图 2-45 所示。那么折半查找不成功的过程就是走了一条从根结点到外部结点的路径。和给定值进行比较的关键字个数等于该路径上内部结点个数。因此折半查找在查找不成功时和给定值进行比较的关键字个数最多也不会超过 $\lfloor \log_2 n \rfloor + 1$ 。

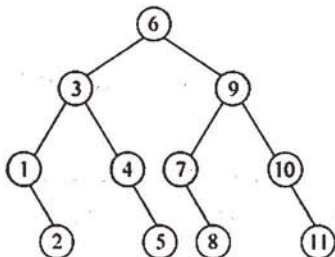


图 2-44 具有 11 个结点的折半查找判定树

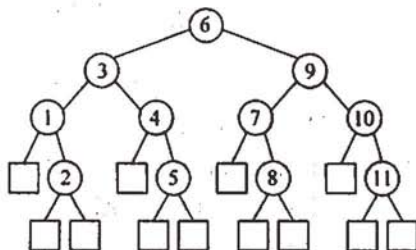


图 2-45 加上外部结点的判定树

那么折半查找的平均查找长度是多少呢? 为了方便起见,不妨设结点总数为 $n=2^h-1$,则判定树是深度为 $h=\log_2(n+1)$ 的满二叉树。在等概率情况下,折半查找的平均查找长度为:

$$ASL_{\text{折半}} = \sum_{i=1}^n P_i C_i = \frac{1}{n} \sum_{j=1}^n j \times 2^{j-1} = \frac{n+1}{n} \log_2(n+1) - 1$$

当 n 值较大时, $ASL_{\text{折半}} \approx \log_2(n+1) - 1$ 。

折半查找比顺序查找的效率要高,但它要求查找表进行顺序存储并且按关键字有序排列。因此,当对表进行元素的插入或删除时,需要移动大量的元素。所以折半查找适用于表轻易不变,且又经常进行查找的情况。

3) 分块查找

(1) 分块查找的基本思想。

分块查找又称索引顺序查找,是对顺序查找方法的一种改进,其性能介于顺序查找与折半查找之间。

在分块查找过程中,首先将表分成若干块,每一块中关键字不一定有序,但块之间是有序的,即后一块中所有记录的关键字均大于前一个块中最大的关键字。此外,还建立了一个“索引表”,索引表按关键字有序。图 2-46 所示的是一个表及其索引表。

因此,查找过程分为两步:第一步在索引表中确定待查记录所在的块;第二步在块内顺序查找。

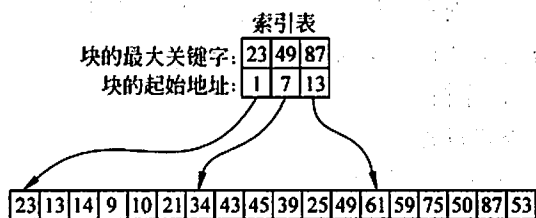


图 2-46 表及其索引表

(2) 分块查找的性能分析。

由于分块查找实际上是两次查找的过程,因此整个分块查找的平均查找长度应该是两次查找的平均查找长度(块内查找与索引查找)之和。所以分块查找的平均查找长度为:

$$ASL_{bs} = L_b + L_w$$

其中, L_b 为查找索引表的平均查找长度, L_w 为块内查找时的平均查找长度。

为了进行分块查找,可以将长度为 n 的表均匀地分成 b 块,每块含有 s 个记录,即有 $b = \lceil \frac{n}{s} \rceil$ 。在等概率的情况下,块内查找的概率为 $\frac{1}{s}$,每块的查找概率为 $\frac{1}{n}$ 。若按顺序查找确定元素所在的块,则分块查找的平均查找长度为:

$$ASL_{bs} = L_b + L_w = \frac{1}{b} \sum_{j=1}^b j + \frac{1}{s} \sum_{i=1}^s i = \frac{b+1}{2} + \frac{s+1}{2} = \frac{1}{2} \left(\frac{n}{s} + s \right) + 1$$

可见在这种条件下其平均查找长度不仅与表长 n 有关,而且和每一块中的记录数 s 有关。可以证明,当 s 取 \sqrt{n} 时, ASL_{bs} 取最小值 $\sqrt{n} + 1$ 。这时的查找性能较顺序查找要好得多,但远不及折半查找。

考虑到索引表是一个有序表,因此可以用折半查找确定元素所在的块。分块查找的平均查找长度为:

$$ASL_{bs} \approx \log_2 \left(\frac{n}{s} + 1 \right) + \frac{s}{2}$$

3. 动态查找表

动态查找表的特点是:表结构本身在查找过程中是动态生成的。即对于给定值 key ,若表中存在关键字等于 key 的记录,则查找成功返回,否则插入关键字等于 key 的记录。

1) 二叉排序树

(1) 二叉排序树的定义。

二叉排序树又称二叉查找树,它或者是一棵空树,或者是具有如下性质的二叉树:

- 若它的左子树非空,则左子树上所有结点的值均小于根结点的值;

- 若它的右子树非空,则右子树上所有结点的值均大于或等于根结点的值;
- 左、右子树本身就是两棵二叉排序树。

图 2-47 为一棵二叉排序树。

(2) 二叉排序树的查找过程。

因为二叉排序树可以看成是一个有序表,在二叉排序树中左子树上所有结点的关键字均小于根结点的关键字,右子树上所有结点的关键字均大于或等于根结点的关键字,所以在二叉排序树上进行查找的过程与折半查找过程类似。

在二叉排序树上进行查找的过程为:若二叉排序树非空,将给定值与根结点的关键字值进行比较。若相等,则查找成功。若不等,则当根结点的关键字值大于给定值时,到根的左子树中进行查找。否则到根的右子树中进行查找。若找到,则查找过程是走了一条从树根到所找到结点的路径。否则,查找过程终止于一棵空树。

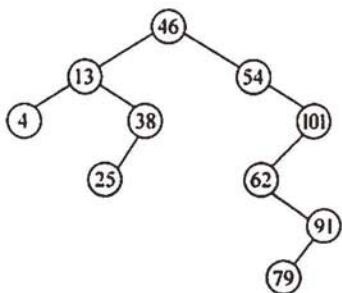


图 2-47 一棵二叉排序树

(3) 二叉排序树中插入结点的操作。

二叉排序树是通过依次输入数据元素并把它们插到二叉树的适当位置上构造起来的。具体的过程是:每读入一个元素,即建立一个新结点。若二叉排序树非空,则将新结点的值与根结点的值进行比较。如果小于根结点的值,则插入左子树中,否则插入右子树中。若二叉排序树为空,则新结点作为二叉排序树的根结点。设关键字序列为{46,25,54,13,29,91},则整个二叉排序树的构造过程如图 2-48 所示。

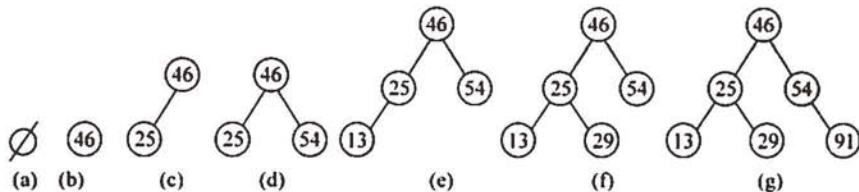


图 2-48 二叉排序树的构造过程

从上面的插入过程还可以看到,每次插入的新结点都是二叉排序树上新的叶子结点,则在插入操作时,不必移动其他结点,仅须改动某个结点的指针,使其由空变为非空即可。这就相当于在一个有序序列上插入一个记录而不需要移动其他记录。它表明,二叉排序树具有类似于折半查找的特性,可采用链表作存储结构,因此是动态查找表的一种适宜表示。

另外,由于一棵二叉排序树的形态完全由输入序列决定,在输入序列已经有序的情况下,所构造的二叉排序树是一棵单枝树。从二叉排序树的查找过程可知,这种情况下的查找效率和顺序查找的效率是相同的。

(4) 二叉排序树中删除结点的操作。

在二叉排序树中删除一个结点时,不能把以该结点为根的子树都删除,只能删除这个结点并仍旧保持二叉排序树的特性。也就是说,删除二叉排序树上一个结点相当于删除有序序列中的一个元素。

假设在二叉排序树上被删除结点为 *p(p 指针指向被删除结点), *f 为其双亲结点,则删除结点 *p 的过程可分为 3 种情况。

- 若 *p 结点为叶子结点,即 $p \rightarrow lchild$ 及 $p \rightarrow rchild$ 均为空,则由于删去叶子结点后不破坏整棵树的结构,因此只须修改 *p 结点的双亲结点 *f 的相应指针即可

$f \rightarrow lchild$ (或 $f \rightarrow rchild$) = NULL;

- 若 *p 结点只有左子树或者只有右子树,此时只要将 *p 的左子树或右子树置为其双亲结点 *f 的左子树(或右子树),即令

$f \rightarrow lchild$ (或 $f \rightarrow rchild$) = $p \rightarrow lchild$; 或 $f \rightarrow lchild$ (或 $f \rightarrow rchild$) = $p \rightarrow rchild$;

- 若 *p 结点的左、右子树均不空,此时不能像上面那样简单处理。删除 *p 结点时应将 *p 的左子树、右子树连接到适当的位置,并保持二叉排序树的特性。可采用如下两种方法进行处理:一是令 *p 的左子树为 *f 的左(或右)子树,而将 *p 的右子树下接到 *p 的中序遍历的直接前驱结点 *s(*s 结点是 *p 的左子树中最右边的结点)的右孩子指针上;二是用 *p 的中序直接前驱(或后继)结点 *s 代替 *p 结点,然后删除 *s 结点。

从二叉排序树的定义可知,中序遍历二叉排序树可得到一个关键字有序的序列。这也说明,一个无序序列可以通过构造一棵二叉排序树而变成一个有序序列。构造树的过程即为对无序序列进行排序的过程。

2) 平衡二叉树

平衡二叉树又称为 AVL 树,它或者是一棵空树,或者是具有下列性质的二叉树:它的左子树和右子树都是平衡二叉树,且左子树和右子树的深度之差的绝对值不超过 1。若将二叉树结点的平衡因子(Balance Factor, BF)定义为该结点的左子树的深度减去其右子树的深度,则平衡二叉树上所有结点的平衡因子只可能是 -1、0 和 1。只要树上有一个结点的平衡因子的绝对值大于 1,则该二叉树就是不平衡的。

分析二叉排序树的查找过程可知,只有在树的形态比较均匀的情况下,查找性能才能达到最佳。因此,在构造二叉排序树的过程中,希望它能保持为一棵平衡二叉树。

使二叉排序树保持平衡的基本思想是：每当在二叉排序树中插入一个结点时，首先要检查是否因插入而破坏了平衡。若是，则找出其中的最小不平衡二叉树，在保持二叉排序树特性的情况下，调整最小不平衡子树中结点之间的关系，以达到新的平衡。所谓最小不平衡子树指离插入结点最近且以平衡因子的绝对值大于1的结点作为根的子树。

(1) 平衡二叉树上的插入操作。

一般情况下，假设由于在二叉排序树上插入结点而失去平衡的最小子树根结点的指针为 a ，也就是说， a 所指结点是离插入结点最近且平衡因子的绝对值超过1的祖先结点。那么，失去平衡后进行调整的规律可归纳为以下4种情况。

- 单向右旋平衡处理：由于在 $*a$ 的左子树根结点的左子树上插入新结点， $*a$ 的平衡因子由1增至2，导致以 $*a$ 为根的子树失去平衡，所以需要进行一次向右的顺时针旋转操作，如图2-49所示。

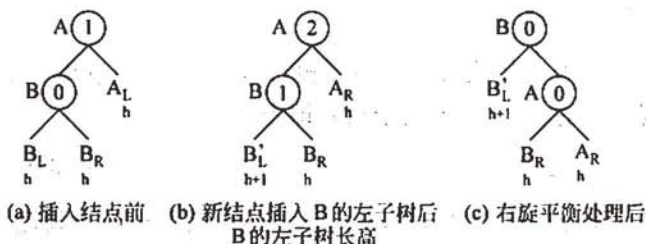


图 2-49 单向右旋平衡处理示意图

- 单向左旋平衡处理：由于在 $*a$ 的右子树根结点的右子树上插入新结点， $*a$ 的平衡因子由-1变为-2，导致以 $*a$ 为根的子树失去平衡，所以需进行一次向左的逆时针旋转操作，如图2-50所示。

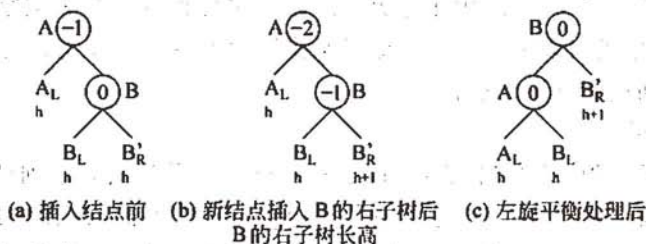
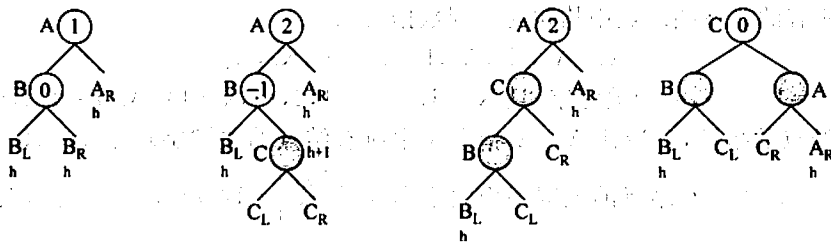


图 2-50 单向左旋平衡处理示意图

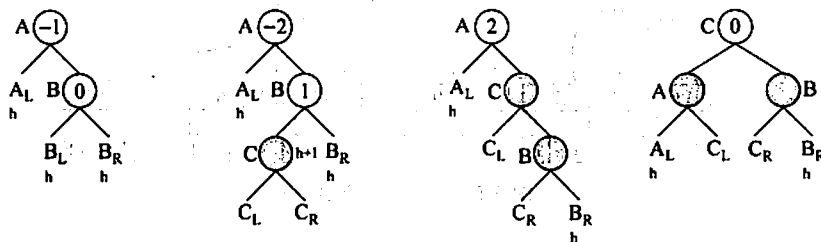
- 双向旋转(先左后右)平衡处理：由于在 $*a$ 的左子树根结点的右子树上插入新结点， $*a$ 的平衡因子由1增至2，致使以 $*a$ 为根结点的子树失去平衡，所以需要进行两次旋转(先左旋后右旋)操作，如图2-51所示。



(a) 插入结点前 (b) 新结点插入 B 的右子树后 B 的右子树长高 (c) 左旋平衡处理后 (d) 右旋平衡处理后

图 2-51 先左后右双向平衡处理示意图

- 双向旋转(先右后左)平衡处理：由于在 * a 的右子树根结点的左子树上插入新结点，* a 的平衡因子由 -1 变为 -2，致使以 * a 为根结点的子树失去平衡，则需要进行两次旋转(先右旋后左旋)操作，如图 2-52 所示。



(a) 插入结点前 (b) 新结点插入 B 的左子树后 B 的左子树长高 (c) 右旋平衡处理后 (d) 左旋平衡处理后

图 2-52 先右后左双向平衡处理示意图

(2) 平衡二叉树上的删除操作。

在平衡二叉树上进行删除操作比插入操作更复杂。若待删结点的两个子树都不为空，就用该结点左子树上中序遍历的最后一个结点(或其右子树上的第一个结点)替换该结点，使之转化为待删除结点只有一个子树后再进行处理。当一个结点被删除后，从被删结点到树根的路径上所有结点的平衡因子都需要更新。对于每一个位于该路径上的平衡因子为 ± 2 的结点来说，都要进行平衡处理。

3) B 树定义

一棵 m 阶的 B-树，或为空树，或为满足下列特性的 m 叉树。

- (1) 树中每个结点至多有 m 棵子树；
- (2) 若根结点不是叶子结点，则至少有两棵子树；
- (3) 除根之外的所有非终端结点至少有 $\lceil \frac{m}{2} \rceil$ 棵子树；

(4) 所有的非终端结点中均包含下列数据信息:

$$(n, A_0, K_1, A_1, K_2, A_2, \dots, K_n, A_n)$$

其中, $K_i (i=1, 2, \dots, n)$ 为关键字, 且 $K_i < K_{i+1} (i=1, 2, \dots, n-1)$, $A_i (i=0, 1, \dots, n)$ 为指向子树根结点的指针, 且指针 A_{i-1} 所指子树中所有结点的关键字均小于 $K_i (i=1, 2, \dots, n)$, A_n 所指子树中所有结点的关键字均大于 K_n , $n \left(\left\lfloor \frac{m}{2} \right\rfloor - 1 \leq n \leq m-1 \right)$ 为结点中关键字的个数。

(5) 所有的叶子结点都出现在同一层次上, 并且不带信息(可以看作是外部结点或查找失败的结点, 实际上这些结点不存在, 指向这些结点的指针为空)。

一棵 4 阶的 B-树如图 2-53 所示。

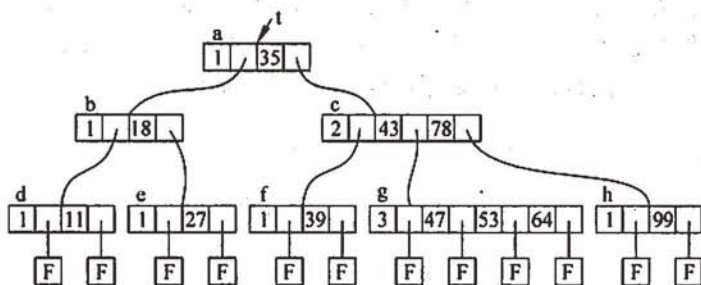


图 2-53 一棵 4 阶 B-树

由 B-树的定义可知, 在 B-树上进行查找的过程是: 首先在根结点包含的关键字中查找给定的关键字, 若找到则成功返回; 否则确定待查找的关键字所在的子树并继续进行查找, 直到查找成功或查找失败(指针为空)时为止。

B-树上的插入和删除运算较为复杂, 因为要保证运算后结点中关键字的个数大于等于 $\left\lfloor \frac{m}{2} \right\rfloor - 1$, 因此涉及到结点的“分裂”及“合并”问题。

在 B-树中插入一个关键字时, 不是在树中增加一个叶子结点, 而是首先在低层的某个非终端结点中添加一个关键字, 若该结点中关键字的个数不超过 $m-1$, 则完成插入。否则, 要进行结点的“分裂”处理。所谓“分裂”, 就是把结点中处于中间位置上的关键字取出来, 插到其父结点中, 并以该关键字为分界线, 把原结点分成两个结点。“分裂”过程可能会一直持续到树根。

同样, 在 B-树中删除一个结点时, 首先找到关键字所在的结点, 若该结点在含有信息的最后一层, 且其中关键字的数目不少于 $\left\lfloor \frac{m}{2} \right\rfloor - 1$, 则完成删除; 否则需进行结点的“合并”运算。若待删除的关键字所在结点不在含有信息的最后一层上, 则将该关键字用其在 B-树中的后继替代, 然后再删除其后继元素, 即将需要处理的情况统一转化为在含

有信息的最后一层再进行删除运算。

4. 哈希表及其查找

1) 哈希表的定义

根据设定的哈希函数 $H(key)$ 和处理冲突的方法, 将一组关键字映射到一个有限的连续地址集(区间)上, 并以关键字在地址集中的“像”作为记录在表中的存储位置, 这种表称为哈希表。这一映射过程称为哈希造表或散列, 所得的存储位置称为哈希地址或散列地址。

前面讨论的几种查找方法, 由于记录在存储结构中的相对位置是随机的, 所以在查找时都要通过一系列关键字的比较才能确定被查记录在结构中的位置。也就是说, 这类查找都是以关键字的比较为基础的, 而哈希表则是通过一个以记录的关键字为自变量的函数(称为哈希函数)得到该记录的存储地址而构造的查找表, 所以在哈希表中进行查找操作时, 可以用同一哈希函数计算得到待查记录的存储地址, 然后到相应的存储单元里获得有关信息再判定查找是否成功。

对于某个哈希函数 H 和两个关键字 K_1 和 K_2 , 如果 $K_1 \neq K_2$, 而 $H(K_1) = H(K_2)$, 这种现象称为冲突。具有相同函数值的关键字对该哈希函数来说称为同义词。

一般情况下, 冲突只能尽可能减少, 而不能完全避免。因为哈希函数是从关键字集合到地址集合的映像。通常关键字集合比较大, 它的元素包含所有可能的关键字, 而地址集合的元素仅为哈希表中的地址值。设关键字序列为某种高级语言的所有标识符, 如果一个标识符对应一个存储地址, 那就不会发生冲突了, 但这是不可能也没有必要的, 因为存储空间难以满足, 而且任何一个源程序不会有这么多标识符。因此在一般情况下, 哈希函数是一个压缩映像, 冲突是不可避免的。所以在建造哈希表时不仅要设定一个“好”的哈希函数, 而且要设定一种处理冲突的方法。

对于哈希表, 主要考虑两个问题: 其一是如何构造哈希函数, 其二是如何解决冲突。对于哈希函数的构造, 应解决好两个主要问题:

- (1) 哈希函数应是一个压缩映像函数, 它应具有较大的压缩性, 以节省存储空间;
- (2) 哈希函数应具有较好的散列性, 虽然冲突是不可避免的, 但应尽量减少。

2) 哈希函数的构造方法

要减少冲突, 就要设法使哈希函数尽可能均匀地把关键字映射到符号表存储区的各个存储地址上, 这样就可以提高查找效率。构造哈希函数时, 一般都要对关键字进行计算。为了尽量避免产生相同的哈希函数值, 应使关键字的所有组成部分都能起作用。

常用的哈希函数构造方法有直接定址法、数字分析法、平方取中法、折叠法、随机数法和除留余数法等。

3) 处理冲突的方法

解决冲突就是为出现冲突的关键字找到另一个“空”的哈希地址。在处理冲突的过程中可能得到一个地址序列 $H_i (i=1, 2, \dots, k)$ 。常见的冲突处理方法如下所述。

(1) 开放地址法。

$$H_i = (H(key) + d_i) \% m \quad i=1, 2, \dots, k \quad (k \leq m-1)$$

其中, $H(key)$ 为哈希函数, m 为哈希表表长, d_i 为增量序列, d_i 有 3 种取法: $d_i=1, 2, \dots, m-1$, 称为线性探测再散列; $d_i=1^2, -1^2, 2^2, -2^2, \dots, \pm k^2 (k \leq m/2)$, 称为二次探测再散列; d_i 为伪随机序列, 称为随机探测再散列。

产生探测序列最简单的方法是进行线性探测, 即当发生冲突时, 顺序地到存储区的下一个单元进行探测。例如, 某记录的关键字为 key , 哈希函数值 $H(key)=j$, 若在 j 位置上发生冲突, 则顺序地对 $j+1$ 位置进行探测。依此类推, 将元素存入哈希表。在查找时也有 3 种可能: 第一种情况是在某一位置上查到了关键字等于 key 的记录, 查找成功; 第二种情况是按探测序列查不到关键字为 key 的记录而又遇到了空单元, 这就可以进行插入操作; 第三种情况是查遍全表, 未查到指定关键字且符号表存储区已满, 需进行溢出处理。

线性探测法思路清楚, 算法简单, 但也存在以下缺点。

- 溢出处理需另编程序: 一般可另外设立一个溢出表, 专门用来存放上述哈希表中放不下的记录。实现溢出表的最简单的结构是顺序表, 查找方法可用顺序查找。
- 线性探测法很容易产生聚集现象: 所谓聚集现象就是存入哈希表的记录在表中连成一片。当哈希函数不能把关键字很均匀地散列到哈希表中时, 尤其容易产生聚集现象。产生聚集现象后, 会增加探测的次数, 从而降低了查找效率。

可以采取多种方法减少聚集现象的产生, 二次探测再散列和随机探测再散列是两种有效的方法。

(2) 链地址法。

链地址法是一种经常使用且很有效的方法。它在符号表的每一个记录中增加一个链域, 链域中存放下一个具有相同哈希函数值的记录的存储地址。利用链域, 就把若干个发生冲突的记录连接在一个链表内。当链域的值为 NULL 时, 表示已没有后继记录了。因此, 发生冲突时的查找和插入操作就跟线性表一样了。

(3) 再哈希法。

$$H_i = RH_i(key) \quad (i=1, 2, \dots, k)$$

RH_i 是不同的哈希函数, 即在同义词发生地址冲突时利用另一个哈希函数计算地址, 直到冲突不再发生。这种方法不易产生聚集现象, 但增加了计算时间。

(4) 建立一个公共溢出区。

不管由哈希函数得到的哈希地址是什么, 一旦发生冲突, 都填入公共溢出区中。

4) 哈希表的查找及其性能分析

从哈希表的查找过程可知:

(1) 虽然哈希表在关键字与记录的存储位置之间建立了直接映像,但由于“冲突”的产生,使得哈希表的查找过程仍然是一个给定值和关键字进行比较的过程。因此,仍须以平均查找长度衡量哈希表的查找效率。

(2) 查找过程中须与给定值进行比较的关键字的个数取决于下列 3 个因素: 哈希函数、处理冲突的方法和哈希表的装填因子。

在一般情况下,冲突处理方法相同的哈希表,其平均查找长度依赖于哈希表的装填因子。哈希表的装填因子定义为:

$$\alpha = \frac{\text{表中装入的记录数}}{\text{哈希表的长度}}$$

α 表示哈希表的装满程度。直观地看, α 越小,发生冲突的可能性就越小。反之, α 越大,表中已填入的记录越多,再填记录时,发生冲突的可能性就越大,查找时,给定值需与之进行比较的关键字的个数也就越多。

2.1.6 排序

1. 排序的基本概念及运算

- 排序: 假设含 n 个记录的文件内容为 $\{R_1, R_2, \dots, R_n\}$, 其相应的关键字为 $\{k_1, k_2, \dots, k_n\}$ 。经过排序确定一种排列 $\{R_{j_1}, R_{j_2}, \dots, R_{j_n}\}$, 使得它们的关键字满足如下递增(或递减)关系: $k_{j_1} \leq k_{j_2} \leq \dots \leq k_{j_n}$ (或 $k_{j_1} \geq k_{j_2} \geq \dots \geq k_{j_n}$), 这样的运算称为排序。若在待排序的一组序列中, k_i 和 k_j 的关键字相同, 即 $k_i = k_j$, 且在排序前 k_i 领先于 k_j , 那么当排序后, 如果 k_i 和 k_j 的相对次序保持不变, k_i 仍领先于 k_j , 则称此类排序方法为稳定的。若在排序后的序列中有可能出现 k_j 领先于 k_i 的情形, 则称此类排序为不稳定的。
- 内部排序: 指待排序记录全部存放在内存中进行排序的过程。
- 外部排序: 指待排序记录的数量很大, 以至内存不能容纳全部记录, 在排序过程中尚需对外存进行访问的排序过程。

在排序过程中须进行下列两种基本操作:

- (1) 比较两个关键字的大小;
- (2) 将记录从一个位置移动到另一个位置。前一种操作对大多数排序方法来说都是必要的, 而后一种操作可以通过改变记录的存储方式来予以避免。

2. 简单排序

1) 直接插入排序

直接插入排序是一种简单的排序方法, 具体做法是: 在插入第 i 个记录时, R_1, R_2, \dots, R_{i-1} 已经排好序, 这时将关键字 k_i 依次与关键字 $k_{i-1}, k_{i-2}, \dots, k_1$ 进行比较, 从而找

到应该插入 k_i 的位置。插入元素时,插入位置及其后的记录依次向后移动。

直接插入排序的过程如图 2-54 所示。

[初始关键字]:	[48]	37	64	96	75	12	26	<u>48</u>
$i=2$:	(37)	[37 48]	64	96	75	12	26	<u>48</u>
$i=3$:	(37)	[37 48 64]	96	75	12	26	<u>48</u>	
$i=4$:	(37)	[37 48 64 96]	75	12	26	<u>48</u>		
$i=5$:	(75)	[37 48 64 75 96]	12	26	<u>48</u>			
$i=6$:	(12)	[12 37 48 64 75 96]	26	<u>48</u>				
$i=7$:	(26)	[12 26 37 48 64 75 96]	<u>48</u>					
$i=8$:	(48)	[12 26 37 48 64 75 96]						

↑ 监视哨 r[0]

图 2-54 直接插入排序示例

2) 冒泡排序

对 n 个记录进行冒泡排序的方法是: 首先将第一个记录的关键字和第二个记录的关键字进行比较。若为逆序,则交换两个记录的值,然后比较第二个记录和第三个记录的关键字。依次类推,直至对第 $n-1$ 个记录和第 n 个记录的关键字进行比较为止。上述过程称作第一趟冒泡排序,其结果是关键字最大的记录被安置到第 n 个记录的位置上。然后进行第二趟冒泡排序,对前 $n-1$ 个记录进行同样的操作,其结果是关键字次大的记录被安置到第 $n-1$ 个记录的位置上。当进行完第 $n-1$ 趟排序时,所有记录已按序排列。

【算法】冒泡排序算法。

```
void bubblesort(int data[],int n){
/* 将数组 data 中的 n 个整数按非递减有序的方式进行排列 */
int i,j,tag,temp;
for(i=0,tag=1;tag==1&& i<n-1;i++){
    tag = 0;
    for(j=0;j<n-i-1;j++){
        if (data[j]>data[j+1]){
            temp=data[j]; data[j]=data[j+1]; data[j+1]=temp;
            tag=1;
        }/* if */
    }/* for */
}/* bubblesort */
```

3) 简单选择排序

对 n 个记录进行简单选择排序的基本方法是: 通过 $n-i$ 次关键字之间的比较,从 $n-i+1$ 个记录中选出关键字最小的记录,并和第 i ($1 \leq i \leq n$) 个记录进行交换。当 i 等于

n 时所有记录已按序排列。

【算法】简单排序。

```
void selectsort(int data[],int n){
/* 将数组 data 中的 n 个整数按非递减有序的方式进行排列 */
int i,j,k,temp;
for(i=0;i<n-1;i++){
    k=i;
    for(j=i+1;j<n;j++) /* 找出最小关键字的下标 */
        if (data[j]<data[k]) k=j;
    if (k!=i) {
        temp=data[i]; data[i]=data[k]; data[k]=temp;
    } /* if */
} /* for */
} /* selectsort */
```

3. 希尔排序

希尔排序又称“缩小增量排序”，是对直接插入排序方法的改进。

希尔排序的基本思想：先将整个待排序记录序列分割成若干序列，然后分别进行直接插入排序。待整个序列中的记录基本有序时，再对全体记录进行一次直接插入排序。具体做法是：先取定一个小于 n 的整数 d_1 作为第一个增量，把文件的全部记录分成 d_1 个组，将所有距离为 d_1 倍数的记录放在同一个组中，在各组内进行直接插入排序。然后取第二个增量 $d_2 < d_1$ ，重复上述分组和排序工作。依此类推，直至所取的增量 $d_i = 1$ ($d_i < d_{i-1} < \dots < d_2 < d_1$)，即将所有记录放在同一组进行直接插入排序为止。

希尔插入排序过程如图 2-55 所示。

[初始关键字]:	48	37	64	96	75	12	26	48	54	03
	48					12				
		37					26			
			64					48		
				96					54	
					75					03
第一趟排序结果:	12	26	48	54	03	48	37	64	96	75
	12			54			37			75
		26			03			64		
			48			48			96	
第二趟排序结果:	12	03	48	37	26	48	54	64	96	75
第三趟排序结果:	03	12	26	37	48	48	54	64	75	96

图 2-55 希尔排序示例

【算法】用希尔排序方法对整型数组进行非递减排序。

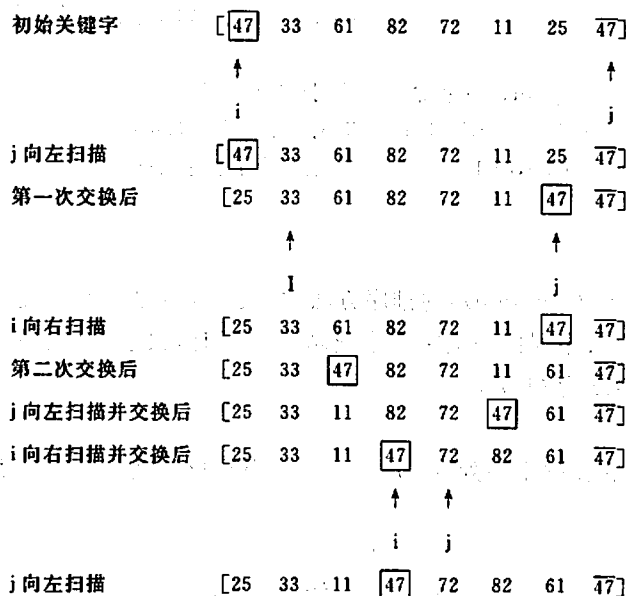
```
void shellsort(int data[],int n){
    int * delta,k,i,t,dk,j;
    k = n;
    /* 从 k=n 开始,重复 k=k/2 运算,直到 k 等于 0。所得 k 值的序列作为增量序列存入 delta */
    delta = (int *)malloc(sizeof(int) * (n/2));
    i = 0;
    do{
        k = k/2;    delta[i++] = k;
    }while(k>0);
    i = 0;
    while (dk=delta[i]>0) {
        for(k = delta[i];k < n; ++k)
            if (data[k]<data[k-dk]) { /* 将元素 data[k]插入有序增量子表中 */
                t = data[k];          /* 备份待插入的元素,空出一个元素位置 */
                for(j = k-dk;j>=0&& t<data[j];j-=dk)
                    data[j+dk]=data[j]; /* 寻找插入位置的同时元素后移 */
                data[j+dk]=t;          /* 找到插入位置,插入元素 */
            } /* if */
        ++i; /* 取下一个增量值 */
    } /* while */
} /* shellsort */
```

4. 快速排序

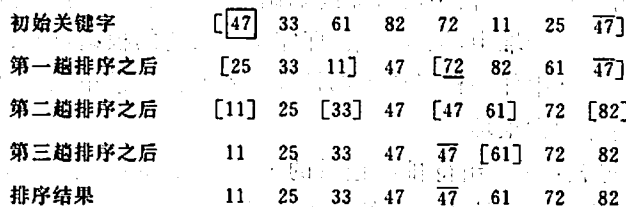
快速排序的基本思想：通过一趟排序将待排序的记录分割为独立的两部分，其中一部分记录的关键字均比另一部分记录的关键字小，然后再分别对这两部分记录继续进行排序，以达到整个序列有序。

一趟快速排序的具体做法是：附设两个指针 low 和 high，它们的初值分别指向文件的第一个记录和最后一个记录。设枢轴记录（通常是第一个记录）的关键字为 pivotkey，则首先从 high 所指位置起向前搜索，找到第一个关键字小于 pivotkey 的记录并与枢轴记录互相交换。然后从 low 所指位置起向后搜索，找到第一个关键字大于 pivotkey 的记录并与枢轴记录互相交换。重复这两步直至 low=high 为止。

快速排序的过程如图 2-56 所示。



(a) 第一趟扫描过程



(b) 各趟排序之后的状态

图 2-56 快速排序示例

【算法】用快速排序方法对整型数组进行非递减排序。

```
void quicksort(int data[],int low,int high){
/* 用快速排序方法对数组元素 data[low..high]作非递减排序 */
int i,pivot,j;
if (low<high) { /* 以数组的第一个元素为基准进行划分 */
    pivot = data[low]; i = low; j = high;
    while(i<j) { /* 从数组的两端交替地向中间扫描 */
        while(i<j && data[j]>=pivot) j--;
        if (i<j) data[i++] = data[j]; /* 比枢轴元素小者移到低下标端 */
        while (i<j && data[i]<=pivot) i++;
    }
    data[i] = pivot;
    quicksort(data,low,i-1);
    quicksort(data,i+1,high);
}
```

```

    if (i < j) data[j--] = data[i]; /* 比枢轴元素大者移到高下标端 */
  }
  data[i] = pivot; /* 枢轴元素移到正确的位置 */
  quicksort(data, low, i-1); /* 对前半个子表递归排序 */
  quicksort(data, i+1, high); /* 对后半个子表递归排序 */
} /* if */
} /* quicksort */

```

在所有同数量级($O(n \log n)$)的排序方法中,快速排序法被认为是平均性能最好的一种。但是,若初始记录序列按关键字有序或基本有序时,快速排序将蜕化为冒泡排序,此时,算法的时间复杂度为 $O(n^2)$ 。

5. 堆排序

对于 n 个元素的关键字序列 $\{k_1, k_2, \dots, k_n\}$, 当且仅当所有关键字都满足下列关系时称其为堆:

$$\begin{cases} k_i \leq k_{2i} \\ k_i \geq k_{2i+1} \end{cases} \text{ 或 } \begin{cases} k_i \leq k_{2i} \\ k_i \geq k_{2i+1} \end{cases}$$

若将此序列对应的一维数组(即以一维数组作为序列的存储结构)看成一个完全二叉树,则堆的含义表明,完全二叉树中所有非终端结点的值均不大于(或不小于)其左、右孩子结点的值。因此,在一个堆中,堆顶元素(即完全二叉树的根结点)必为序列中的最小元素(或最大元素),并且堆中任一棵子树也都是堆。若堆顶为最小元素,则称为小根堆。若堆顶为最大元素,则称为大根堆。

堆排序的基本思想:对一组待排序记录的关键字,首先把它们按堆的定义排成一个推序列(即建立初始堆),从而输出堆顶的最小关键字(对于小顶堆而言)。然后将剩余的關鍵字再调整成新堆,以便得到次小的关键字。如此反复进行,直到全部关键字排成有序序列为止。

初始堆的建立方法是:将待排序的关键字分放到一棵完全二叉树的各个结点中(此时完全二叉树并不一定具备堆的特性),显然,所有 $i > \lfloor n/2 \rfloor$ 的结点 K_i 都没有子结点。以这样的 K_i 为根的子树已经是堆,因此初始建堆可从完全二叉树的第 $i (i = \lfloor n/2 \rfloor)$ 个结点 K_i 开始,通过调整,逐步使以 $K_{\lfloor n/2 \rfloor}, K_{\lfloor n/2 \rfloor - 1}, K_{\lfloor n/2 \rfloor - 2}, \dots, K_2, K_1$ 为根的子树满足堆的定义。在对以 K_i 为根的子树建堆的过程中,可能需要交换 K_i 和 $K_{2 \times i}$ (或 $K_{2 \times i + 1}$) 的值。如此,以 $K_{2 \times i}$ (或 $K_{2 \times i + 1}$) 为根的子树可能不再满足堆的定义,此时则应继续以 $K_{2 \times i}$ (或 $K_{2 \times i + 1}$) 为根进行调整。如此层层地递推下去,可能会一直延伸到树叶时为止。这种方法就像过筛子一样,把最小的关键字一层一层地筛选出来,最后输出堆顶的最小元素。

【算法】 将一个整型数组中的元素调整成大根堆。


```

void heapadjust(int data[],int s, int m) {
/* data[s..m]构成的元素序列中,除了 data[s]外,其余元素均满足堆的定义 */
/* 调整元素 data[s]的位置,使 data[s..m]成为一个大根堆 */
int t,j;
t=data[s];          /* 备份元素 data[s],为其找到适当位置后再插入 */
for(j=2*s+1;j<=m;j=j*2+1) { /* 沿着值较大的孩子结点方向向下筛选 */
    if (j<m && data[j]<data[j+1])
        ++j; /* j 是值较大的元素的下标 */
    if (! (t<data[j]))
        break;
    data[s]=data[j];s=j; /* 用 s 记录待插入元素的位置(下标) */
} /* for */
data[s]=t; /* 将备份元素插入由 s 所指出的插入位置 */
} /* heapadjust */

```

调整成新堆：假设输出堆顶元素之后，以堆中最后一个元素替代之，那么根结点的左、右子树均为堆，此时只需自上至下进行调整即可。

【算法】用堆排序方法对整型数组进行非递减排序。

```

void heapsort(int data[],int n){ /* 对数组 data[0..n-1]中的 n 个元素进行堆排序 */
register int i;
int t;
for(i=n/2-1;i>=0;--i) /* 把 data[0..n-1]调整为大根堆 */
    heapadjust(data,i,n-1);
for(i=n-1;i>0;--i) {
    t=data[0];
    data[0]=data[i]; /* 堆顶元素 data[0]与序列的最后元素 data[i]交换 */
    data[i]=t;
    /* 待排序元素的个数减 1,将 data[0..i-1]重新调整为大根堆 */
    heapadjust(data,0,i-1);
}
}

```

初始堆建立过程如图 2-57 所示,调整为新堆的过程图 2-58 所示。

对于记录数较少的文件来说,堆排序的优越性并不明显,但对大量的记录来说,堆排序是有效的。堆排序的整个算法时间是由建立堆和不断调整堆这两部分时间代价构成的。

可以证明,堆排序算法的时间复杂度为 $O(n\log n)$ 。此外,堆排序只需要一个记录大

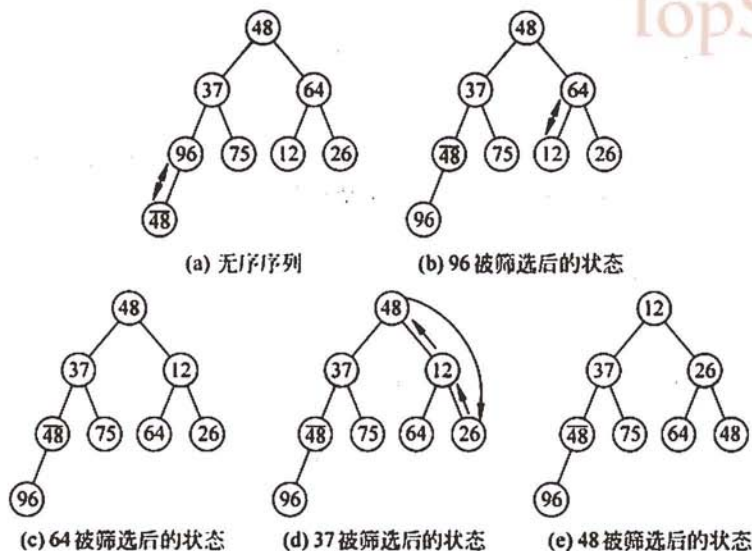


图 2-57 初始堆建立过程示例

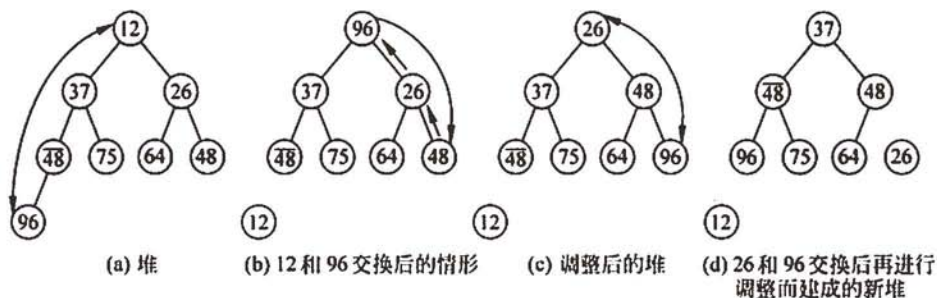


图 2-58 调整为新堆的过程

小的辅助空间。堆排序是一种不稳定的排序方法。

6. 归并排序

所谓“归并”是将两个或两个以上的有序文件合并为一个新的有序文件。从线性表的讨论可知,将两个有序表合并为一个有序表,无论顺序存储结构还是链式存储结构,都是容易实现的。利用归并的思想可以进行排序。归并排序是把一个有 n 个记录的无序文件看成由 n 个长度为 1 的有序子文件组成的文件,然后进行两两归并,得到 $\lceil n/2 \rceil$ 个长度为 2 或 1 的有序文件。再两两归并,如此重复,直至最后形成包含 n 个记录的有序文件为止。这种反复将两个有序文件归并成一个有序文件的排序方法称为两路归并排序。

【算法】两路归并。

```
void Merge(int data1[],int data2[],int s,int m,int n){
/* 将分别有序的 data1[s..m]和 data1[m+1..n]归并为有序的 data2[s..n] */
    int i,j,k;
    for(i=m+1,k=s;s<=m && i<=n;++k) /* 将 data1 中记录由小到大并入 data2 */
        if (data1[s]<data1[i])
            data2[k]=data1[s++];
        else data2[k]=data1[i++];
    for(j=s;j<=m;++j,++k) /* 将剩余的 data1[s..m]复制到 data2 */
        data2[k]=data1[j];
    for(j=i;j<=n;++j,++k) /* 将剩余的 data1[i..n]复制到 data2 */
        data2[k]=data1[j];
}/* Merge */
```

【算法】设数组 data1[]中有 n 个元素,可分割为 $k(=\lceil n/\text{len} \rceil)$ 个段,每段分别有序。分段情况如下: data1[0..len-1],data1[len..2*len-1],...,data1[(i-1)*len..i*len-1],...。在下面的函数中,通过调用函数 Merge 将相邻两个分段归并为一个有序段。当调用 $k/2$ 次 Merge 后,两两归并的结果存放在 data2[]中。

```
void Msort(int data1[],int data2[],int n,int len){
    int start_p,end_p;
    start_p=0; /* 从下标为 0 的数组元素开始 */
    while (start_p+len<n) { /* 每次取相邻的两个有序分段进行合并 */
        end_p=start_p+2*len-1;
        if (end_p>=n) end_p=n-1; /* 若仅剩余一个有序分段,退出 */
        Merge(data1,data2,start_p,start_p+len-1,end_p);
        /* 调用 merge 将两个有序段合并为一个有序段,合并后的结果在 data2 中 */
        start_p=end_p+1;
    }/* while */
    if (start_p<n) /* 直接将剩下的一个有序段中的元素复制到 data2 中 */
        for(;start_p<n;start_p++) data2[start_p]=data1[start_p];
}/* Msort */
```

【算法】对数组 data1[0..n-1]中的 n 个元素进行归并排序。首先,数组中的每个元素自成一个长度为 1 的有序分段,共有 n 个分段。然后,将有序分段两两合并,使得分段数目逐渐减少,合并所得分段的长度逐渐加长,当合并到只有一个分段且长度为 n 时,数位元素得到有序排列。

```

void mergesort(int data1[],int n){
    int length=1,k=0;
    int * data2;
    data2=(int *)malloc(sizeof(int) * n);
    if (data2==NULL)
        return;
    /* 第一趟,有序分段在 data1 中,两路归并后,结果存放在 data2 中;第二趟,对存放在 data2 中的
    有序分段进行两路归并,再将结果存放到 data1 中。归并过程在 data1 和 data2 之间交替进行。
    length 用于记录每个分段的长度,k 用于控制交替情况 */
    while(length<n) {
        if (k==0)
            Msort(data1,data2,n,length);
        else
            Msort(data2,data1,n,length);
        length *=2; /* 每次完成一趟归并,所得有序分段的长度翻一倍 */
        k=1-k;
    } /* while */
    if (k==1) /* 若归并趟数为奇数次,排序结果在 data2 中,因此再将结果复制到 data1 中 */
        for(k=0;k<n;k++)
            data1[k]=data2[k];
} /* mergesort */

```

7. 基数排序

基数排序的思想是按组成关键字各个数位的值进行排序的。它是分配排序的一种。

基数排序把关键字 K_i 看成一个 d 元组,即

$$K_i^1, K_i^2, \dots, K_i^d$$

其中, $0 \leq K_i^j < r, i=1 \sim n, j=1 \sim d$ 。这里的 r 称为基数。若关键字是十进制,则 $r=10$ 。若关键字是八进制的,则 $r=8$ 。 d 是关键字的位数。 d 值取所有待排序关键字位数的最大值。其他不足 d 位的关键字则在前面补零。

在 $K_i^1, K_i^2, \dots, K_i^d$ 中, K_i^1 称为最高有效位, K_i^2 称为次高有效位, K_i^d 称为最低有效位。基数排序可以从最高有效位开始,也可以从最低有效位开始。

基数排序的基本思想: 设立 r 个队列, 队列的编号分别为 $0, 1, 2, \dots, r-1$ 。首先按最低有效位的值, 把 n 个关键字分配到这 r 个队列中; 然后从小到大将各队列中的关键字再依次收集起来; 接着再按次低有效位的值把刚收集起来的关键字再分配到 r 个队列中。重复上述收集过程, 直至最高有效位。这样将得到一个从小到大的关键字序列。为了减少记录移动的次数, 队列可以采用链式存储分配, 称为链队列。每个链队列设有两个

指针,分别指向队头和队尾。

分析基数排序算法可知,对于 n 个记录,执行一次分配和收集的时间为 $O(n+r)$ 。如果关键字有 d 位,则要执行 d 遍。所以总的运算时间为 $O(d(n+r))$ 。可见对于不同的基数 r 所用的时间是不同的。当 r 或 d 较小时,这种排序方法较为节省时间。另外,基数排序适用于链式分配的记录的排序,其要求的附加存储量是 r 个队列的头、尾指针,所以附加存储量为 $2r$ 个存储单元。由于待排序记录是以链表方式存储的,故相对于顺序分配而言,还增加了 n 个指针域的空间。基数排序是一种稳定的排序方法。

8. 内部排序方法的比较和选择

综合比较前面讨论的各种排序方法,其结果如表 2-1 所示。

表 2-1 各种排序方法的性能比较

排序方法	最好时间	平均时间	最坏时间	辅助空间	稳定性
直接插入	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
简单选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	—	$O(n^{1.25})$	—	$O(1)$	不稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	$O(rd)$	稳定

迄今为止,已有的排序方法远远不止上述几种,人们之所以热衷于研究各种排序方法,不仅是由于排序在计算机运算中所处的重要位置,而且还因为不同的方法各有优缺点,可根据需要运用到不同的场合。选取排序方法时需要考虑的因素如下:

- (1) 待排序的记录个数 n ;
- (2) 记录本身的大小;
- (3) 关键字的分布情况;
- (4) 对排序稳定性的要求;
- (5) 语言工具的条件,辅助空间的大小。

依据这些因素,可以得到以下几点结论。

(1) 若待排序的记录数目 n 较小时,可采用插入排序和选择排序。由于直接插入排序所需的记录移动操作较直接选择排序多,因而当记录本身信息量较大时,用直接选择排

序方法较好。

(2) 若待排序记录的关键字基本有序,则宜采用直接插入排序或冒泡排序。

(3) 当 n 很大且关键字的位数较少时,采用链式基数排序较好。

(4) 若 n 较大,则应采用时间复杂度为 $O(n \log n)$ 的排序方法:快速排序、堆排序或归并排序。快速排序被认为是目前内部排序方法中最好的方法。当待排序的关键字为随机分布时,快速排序的平均运行时间最短;而堆排序只需 1 处辅助存储空间,并且不会出现在快速排序中可能出现的最坏情况。这两种排序方法都不是稳定排序方法,若要求排序稳定,可选择归并排序。通常可将归并排序和直接插入排序结合起来使用。先利用直接插入排序求得较长的有序子文件,然后再两两归并。因为直接插入排序是稳定的,所以改进的归并排序仍是稳定的。

(5) 本章讨论的内部排序算法(除基数排序外)都是在一维数组上实现的。当记录本身信息量较大时,为避免耗费大量的时间移动记录,可以采用链表作为存储结构。像插入排序和归并排序这类算法都易于在链表上实现,并分别称为表插入和表归并。但有的排序方法,如快速排序和堆排序,不适合在链表结构上实现。在这种情况下,可以提取关键字建立索引表,然后对索引表进行排序。还有一种简单的方法:引入一个整型向量 $t[1..n]$ 作为辅助表,排序前,令 $t[i] = i (1 \leq i \leq n)$;若排序算法中要求交换 $R[i]$ 和 $R[j]$,则只须交换 $t[i]$ 和 $t[j]$ 即可;排序结束后,向量 $t[1..n]$ 就指示了记录之间的顺序关系:

$$R[t[1]].key \leq R[t[2]].key \leq \dots \leq R[t[n]].key$$

如果要求的最终结果是:

$$R[1].key \leq R[2].key \leq \dots \leq R[n].key$$

那么排序结束后,再按链表或辅助表所规定的次序重排各记录,完成这种重排的时间是 $O(n)$ 。

9. 外部排序

外部排序就是对大型文件的排序,待排序的记录存放在外存。在排序过程中,内存只存储文件的一部分记录,整个排序过程需要进行多次的内外存间的数据交换。

常用的外部排序方法是归并排序,这种方法一般分为两个阶段:在第一阶段,把文件中的记录分段读入内存,利用某种内部排序方法对这段记录进行排序并输出到外存的另一个文件中,在新文件中形成许多有序的记录段,称为归并段;在第二阶段,对第一阶段形成的归并段用某种归并方法一趟趟地进行归并,使文件的有序段逐渐加长,直到将整个文件归并为一个有序段时为止。下面简单介绍常用的多路平衡归并方法。

k 路平衡归并是指文件经外部排序的第一个阶段后,已经形成了由若干个初始归并段构成的文件。在这个基础上,反复将每次确定的 k 个归并段归并为一个有序段,将一个文件上的记录归并到另一个文件上。重复这个过程,直到文件中的所有记录都归并为一

个有序段。

设已经得到 8 个初始归并段,如图 2-59 所示,其中 b_i 表示第 i 个归并段。

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7
4	6	28	9	2	26	3	17
17	14	31	21	10	27	22	25
46	55	49	47	52	38	59	33

图 2-59 初始归并段

采用树形选择排序方法对 n 个归并段进行排序,其方法是:首先对 n 个记录的关键字进行两两比较,然后在 $\lceil n/2 \rceil$ 个较小者之间再进行两两比较。如此重复,直到选出关键字最小的记录为止。该过程可用一棵有 n 个叶子结点的完全二叉树表示,如图 2-60(a) 所示。在树中,若每个非终端结点中的关键字等于其左、右孩子结点中较小的关键字,则树根结点中的关键字即为所有叶子中的最小关键字。将最小关键字输出之后,更新最小关键字所在叶子结点的数据,然后从该叶子结点出发,与其左(兄弟)结点的关键字进行比较,依次修改从叶子结点到根的路径上各结点的关键字,直至到达根结点为止。此时根结点中的关键字即为次小关键字,如图 2-60(b) 所示。重复该过程,就可完成对所有记录的排序。

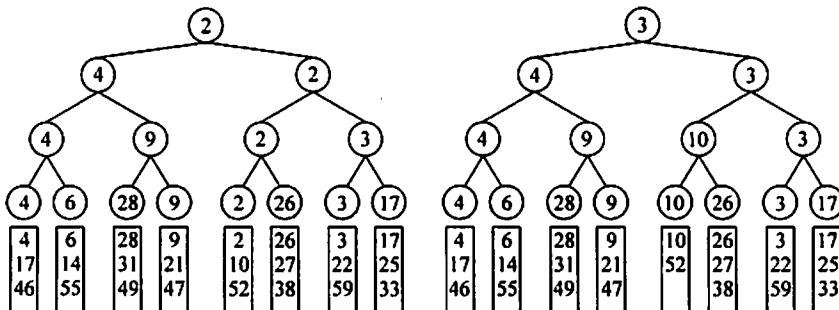


图 2-60 树形选择排序

由于在图 2-60 所示的树中,每个非终端结点记录了其左、右孩子中的“优胜者”,所以我们称其为“胜者树”。反之,若在双亲结点中记录比较后得到的失败者,让胜者参加更上一层的比较,便可得到一棵“败者树”。这样,当优胜者到达父结点时,立刻就知道原先在此比较的失败者并与失败者进行比较,再次记录新的失败者并让优胜者进行更上一层的比较。在败者树中,每个结点只须和其父结点进行比较。而在胜者树中,向上调整时结点需与兄弟结点比较,这就需要得到兄弟结点的位置信息,因此败者树更易于编程。

为了简便起见,设每个记录为一个整数。败者树用数组 $ls[]$ 表示, $ls[i]$ 的值为败者所在归并段的段号。令 $ls[1]$ 为树的根结点, $ls[0]$ 是 $ls[1]$ (根) 的父结点, $ls[0]$ 中存储每次选出的优胜者所在归并段的段号, 输出时则取 $ls[0]$ 指示的归并段的当前记录。例如, 图 2-61 给出的是一棵实现 8 路归并的败者树。其中叶子结点的数据来自各个归并段, b_i 表示第 i 个归并段, 败者树根结点 $ls[1]$ 的父结点 $ls[0]$ 中存储了优胜者 (最小记录) 所在的归并段。图 2-62 所示的是输出一个记录后, 重新调整后的败者树。

【算法】败者树的调整。

```
void Adjust(int ls[K], int s) { /* 从叶子结点  $b[s]$  到根结点的父结点  $ls[0]$  调整败者树 */
    int t, temp;
    t = (s + K) / 2; /*  $t$  为  $b[s]$  的父结点在败者树中的下标,  $K$  是归并段的个数 */
    while (t > 0) { /* 若没有到达树根, 则继续 */
        if (b[s] > b[ls[t]]) { /* 与父结点指示的数据进行比较 */
            /*  $ls[t]$  记录败者所在的段号,  $s$  指示新的胜者。胜者将参加更上一层的比较 */
            temp = s; s = ls[t]; ls[t] = temp;
        } /* if */
        t = t / 2; /* 向树根退一层 */
    } /* while */
    ls[0] = s; /*  $ls[0]$  记录本趟最小关键字所在的段号 */
} /* Adjust */
```

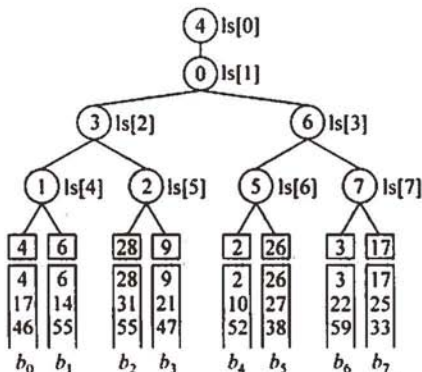


图 2-61 利用败者树找出 8 路归并段的最小关键字

【算法】利用败者树实现 k 路平衡归并。

```
#define K 8
#define MINKEY -1 /* 比所有关键字都小的一个值 */
#define MAXKEY 10000 /* 比所有关键字都大的一个值 */
```

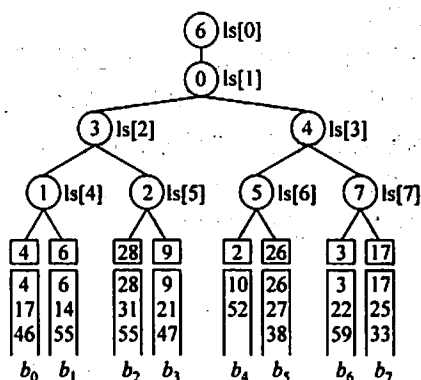



图 2-62 利用败者树找出 8 路归并段的次小关键字

```

void K_merge(int ls[K]){ /* 用败者树实现 K 路平衡归并 */
    /* ls[0]~ls[k-1]是败者树的内部结点 */
    int b[K+1], i, q; /* b[0]~b[k-1]分别存储 K 个初始归并段的当前记录 */
    /* 函数 Get_next(i)用于从第 i 个归并段读取并返回当前记录 */
    /* 若该归并段已空,则置返回值为 MAXKEY */
    for(i=0; i<K; i++) b[i]=Get_next(i); /* 分别读取 K 个归并段的第一个关键字 */
    b[K]=MINKEY;
    for(i=0; i<K; i++) ls[i]=K; /* 设置败者树中结点的初值 */
    for(i=K-1; i>=0; i--) /* 依次从 b[K-1], b[K-2], ..., b[0]出发调整败者 */
        Adjust(ls, i);
    while (b[ls[0]]!=MAXKEY){
        q = ls[0]; /* q 是当前最小关键字所在的归并段 */
        printf("%d", b[q]); /* 第 q 个归并段中的一个元素 */
        b[q]=Get_next(q); /* 从第 q 个归并段读取下一个数据 */
        Adjust(ls, q); /* 调整败者树,选择新的最小关键字 */
    } /* while */
} /* K_merge */

```

2.2 常见算法设计方法

2.2.1 分治法

当需要求解一个输入规模为 n 且取值又相当大的问题时,直接求解是非常困难的,有时甚至无法直接解决。一般的做法是,首先仔细分析问题本身所具有的特征,然后根据这

些特征选择适当的策略来解决。

分治法的思想是：将一个难以直接解决的大问题分解成 k 个子问题，这些子问题与原问题的结构相同，但是规模较小。这时可采用分而治之的方法，首先解决子问题，然后通过合并子问题的解来得到原问题的解。如果分解得到的子问题不能直接求解，则反复使用分治策略将这些子问题分成更小的同类型子问题，直至产生出可直接求解的子问题。由此可知，分治法与递归过程有很自然的对应关系。

分治法的一般设计模式如下：

```

DivideAndConquer (P) {
    if ( |P| ≤ n0 ) adhoc(P);    // |P| 表示问题的规模, n0 为一阈值, adhoc(P) 为基本子算法
    else divide P into smaller subinstances P1, P2, ..., Pk; // 将原问题分解成 k 个子问题
    for (i = 1; i ≤ k; i++)
        yi = DivideAndConquer(Pi);    // 递归求解各个子问题
    return merge(y1, y1, ..., yk);    // 合并子问题的解得到原问题的解
}
    
```

人们从大量实践中发现，在使用分治法设计算法时，最好使子问题的规模大致相同。这种做法出自一种平衡的思想：它几乎总是比子问题规模不等的做法要好。许多问题可以取 $k=2$ 。

【例 2.1】 棋盘覆盖问题。在一个 $2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其他方格不同，称该方格为一特殊方格，该棋盘为一特殊棋盘。显然特殊方格在棋盘上出现的位置有 4^k 种情形。因而对任何 $k \geq 0$ ，有 4^k 种不同的特殊棋盘。在棋盘覆盖问题中，要求使用图 2-63 所示的四种不同形态的 L 型骨牌，覆盖给定特殊棋盘上除特殊方格以外的所有方格，且任何两个 L 型骨牌不得重叠覆盖。实际上，在任何一个 $2^k \times 2^k$ 的棋盘覆盖中，用到的 L 型骨牌数恰为 $(4^k - 1)/3$ 。

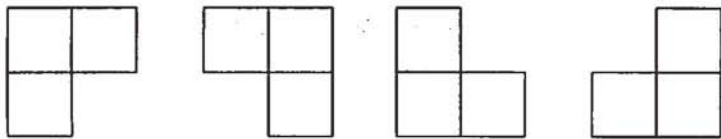


图 2-63 4 种不同形态的 L 形骨牌

使用分治策略，可以设计出解决棋盘覆盖问题的简洁算法。当 $k > 0$ 时，可将 $2^k \times 2^k$ 棋盘分割为 4 个 $2^{k-1} \times 2^{k-1}$ 子棋盘，如图 2-64(a) 所示。

特殊方格必位于 4 个较小子棋盘之一中，其余 3 个子棋盘中无特殊方格。为了将这 3 个无特殊方格的子棋盘转化为特殊棋盘，可以用一个 L 型骨牌覆盖这 3 个较小棋盘的

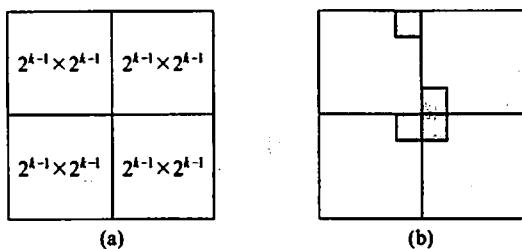


图 2-64 棋盘分割

会合处,如图 2-64(b)所示。这 3 个子棋盘上被 L 型骨牌覆盖的方格就成为该棋盘上的特殊方格,从而将原问题转化为 4 个较小规模的棋盘覆盖问题。递归地使用这种分解技术,直至将棋盘简化为 1×1 棋盘为止。

用分治策略解决棋盘覆盖问题的算法如下:

```
void ChessBoard (int tr, int tc, int dr, int dc, int size){
// tr,tc: 分别表示棋盘左上角方格的行、列号;dr,dc: 分别表示特殊方格所在的行、列号;
// size:  $2^k$ 表示棋盘规格为  $2^k \times 2^k$ ;整型数组 board 表示棋盘,board[0][0]是棋盘的左上角;
// tile 表示 L 型骨牌的编号,初始值为 0。
    if (size==1) return;
    int t = tile++; //L 型骨牌号
    s = size/2; //分割棋盘

//棋盘左上角子棋盘
    if (dr<tr+s && dc<tc+s) //特殊方格在此棋盘中
        ChessBoard(tr,tc,dr,dc,s);
    else { //此棋盘中无特殊方格
        board[tr+s-1][tc+s-1]=t; //用 t 号 L 型骨牌覆盖右下角
        ChessBoard(tr,tc,tr+s-1,tc+s-1,s); //覆盖其余方格
    }

//棋盘右上角子棋盘
    if (dr<tr+s && dc>=tc+s) //特殊方格在此棋盘中
        ChessBoard(tr,tc+s,dr,dc,s);
    else { //此棋盘中无特殊方格
        board[tr+s-1][tc+s]=t; //用 t 号 L 型骨牌覆盖左下角
        ChessBoard(tr,tc+s,tr+s-1,tc+s,s); //覆盖其余方格
    }
}
```

```

//棋盘左下角子棋盘
if (dr >= tr+s && dc < tc+s)    //特殊方格在此棋盘中
    ChessBoard(tr+s,tc,dr,dc,s);
else {    //此棋盘中无特殊方格
    board[tr+s][tc+s-1]=t;    //用 t 号 L 型骨牌覆盖右上角
    ChessBoard(tr+s,tc,tr+s,tc+s-1,s);    //覆盖其余方格
}

//棋盘右下角子棋盘
if (dr >= tr+s && dc >= tc+s)    //特殊方格在此棋盘中
    ChessBoard(tr+s,tc+s,dr,dc,s);
else {    //此棋盘中无特殊方格
    board[tr+s][tc+s]=t;    //用 t 号 L 型骨牌覆盖左上角
    ChessBoard(tr+s,tc+s,tr+s,tc+s,s);    //覆盖其余方格
}

```

2.2.2 动态规划

动态规划法与分治法类似,也是将待求解的问题分解成若干个子问题,先求解子问题,然后从子问题的解得到原问题的解。与分治法不同的是,适合于用动态规划法求解的问题,经分解得到的子问题往往不是独立的。若用分治法来求解,则由于分解得到的子问题太多,而且需要重复求解相同的子问题,使得求解原问题的时间呈幂级数增加。

动态规划法通常用于求解具有最优性质的问题,即问题的最优解包含了其子问题的最优解。这类问题通常有多个解,每个解都对应一个值,我们的目的是找到具有最优值(最大或最小值)的解。为了避免重复求解相同的子问题,引入一个记忆表,用来记录所有已经解决的子问题的解,当再次遇到该子问题时查表即可。

通常可用以下步骤设计动态规划算法:

- ① 找出最优解的性质,并刻画其结构特征;
- ② 递归地定义最优值;
- ③ 以向前处理方式或向后处理方式计算出最优值;
- ④ 根据计算最优值得到的信息,构造最优解。

步骤①~③是动态规划的基本步骤。若要求问题的最优解,则必须执行步骤④。若只需求出最优值,则步骤④可以省略。

【例 2.2】 多边形游戏。多边形游戏是一个单人玩的游戏,开始时有一个由 n 个点构成的多边形。每个顶点赋予一个整数值,每条边赋予一个运算符“+”或“*”。所有的边从 $1 \sim n$ (n 是正整数)进行编号。在游戏的第一步,删除一条边,随后的 $n-1$ 步按以

下方式操作:

- ① 选择一条边 E 以及由 E 连接着的两个顶点 V_1 和 V_2 ;
- ② 用一个新的顶点取代边 E 以及边 E 连接着的两个顶点 V_1 和 V_2 。将由定点 V_1 和 V_2 的整数值通过边 E 上的运算得到的结果赋予新顶点。

当所有边都被删除时游戏结束。游戏的得分就是最后所剩的那个顶点上的数值。问题: 对于给定的多边形, 计算最高得分。

求解多边形游戏问题的动态规划算法如下:

```
void MinMax(int i, int s, int j) {
//从顶点 i 开始, 长度为 j 的顺时针链表示为 p(i, j)。这条链的最后一次合并发生在 op[i+s] 处发生
(1 ≤ s ≤ j-1), 这将把链 p(i, j) 分割为两个子链 p(i, s) 和 p(i+s, j-s)。m1 和 m2 是两个子链的任
意一种合并得到的值, a, b 和 c, d 分别是两个子链合并后得到的最小值和最大值。当 op[i+s]='+'
时, 有 a+c ≤ m ≤ b+d。当 op[i+s]='*' 时, 应有 min{ac, ad, bc, bd} ≤ m ≤ max{ac, ad, bc,
bd}。int []e=new int [5];
int a=m[i][s][0], //m[i][j][0], m[i][j][1] 表示链 p(i, j) 合并后的最小值、最大值
b=m[i][s][1],
r=(i+s-1)%n+1,
c=m[r][j-s][0],
d=m[r][j-s][1];
if (op[r]=='t') { // op[i] 表示第 i 条边所对应的运算符
minf=a+c;
maxf=b+d;
}
else {
e[1]=a*c; e[2]=a*d; e[3]=b*c; e[4]=b*d;
minf=e[1]; maxf=e[1];
for (int k=1; k<5; k++)
if (minf>e[k]) minf=e[k];
if (maxf<e[k]) maxf=e[k];
}
}

int PolyMax() {
for (int j=2; j<=n; j++)
for (int i=1; i<=n; i++)
for (int s=1; s<j; s++) {
MinMax(i, s, j);
if (m[i][j][0]>minf) m[i][j][0]=minf;
```

```

        if (m[i][j][1] < maxf) m[i][j][1] = maxf;
    }
    int temp = m[1][n][1];
    for (int i = 2; i <= n; i++) if (temp < m[i][n][1]) temp = m[i][n][1];
    return temp;
}

```

2.2.3 贪心方法

当一个问题具有最优子结构性时,可用线性规划、整数规划、非线性规划以及动态规划等策略求解。尽管各种规划问题都有一些相应的求解方法,但其中的某些问题还可以用一种更直接、更简单和更有效的方法来求解,这种方法就是贪心方法。

贪心方法的基本思想是:求解过程可分为若干个阶段,在每一个阶段,可以认为所作的决定是最好的(局部最优),而不考虑是否得到全局的最优解。一般来说,这意味着选择的是某个局部的最优。当然,我们也希望用贪心方法得到的最终结果也是整体最优的,因此,选择能产生问题最优解的最优量度标准是使用贪心方法设计求解的核心问题。贪心方法一般的设计模式如下:

```

GreedyMethod(A, int n) {
    //数组 A 包含 n 个输入
    solution ← ∅; //将解向量 solution 初始化为空。
    for (int i = 1; i <= n; i++) {
        x ← select(A); //函数 select 的功能是按最优量度从 A 中选择一个输入,把
        //它的值赋给 x 并从 A 中消去它。
        if feasible(solution, x) solution ← UNION(solution, x);
        //布尔函数 feasible 判定 x 是否可以包含在解向量中,UNION
        //将 x 与解向量结合并修改目标函数。
    }
    return (solution);
}

```

【例 2.3】 二分最小覆盖问题。二分图是一个无向图,它的 n 个顶点被二分为集合 A 和集合 B,且同一集合中的任意两个顶点在图中无边相连(即任何一条边都是一个顶点在集合 A,另一个在集合 B 中)。当且仅当 B 中的每个顶点至少与 A 中一个顶点相连时, A 的一个子集 A' 覆盖集合 B。当且仅当 A' 是覆盖 B 的子集中最小的子集时, A' 为最小覆盖。解决二分最小覆盖问题的贪心算法如下:

```

bool BiPartiteCover(int L[], int C[], int m)

```

(//L 是输入顶点的标号, $L[i]=1$, 当且仅当 i 在集合 A 中。C 是一个记录覆盖的输出数组。如果图中不存在覆盖, 则返回 false, 否则返回 true。在 m 中返回覆盖的大小, 在 $C[0..m-1]$ 中返回覆盖。

```

int n=Vertices(); //顶点数
int SizeofA=0;
for (int i=1; i<=n; i++) //确定集合 A 大小
    if (L[i]==1) SizeofA++;
int SizeofB=n-SizeofA;
CreateBins(SizeofB, n); //创建 SizeofB 个空双向链表, n 个节点
int * New=new int [n+1]; //顶点 i 覆盖了 B 中 New[i] 个未被覆盖的顶点
bool * Change=new bool [n+1]; //Change[i] 为 true, 当且仅当 New[i] 已改变
bool * Cov=new bool [n+1]; //Cov[i] 为 true, 当且仅当 New[i] 被覆盖
InitializePos();
LinkStack <int> S;

for (i=1; i<=n; i++) { //初始化
    Cov[i]=Change[i]=false;
    if (L[i]==1) { //i 在 A 中
        New[i]=Degree(i); //这些皆被覆盖
        InsertBins(New[i], i); //在链表 New[i] 中添加结点 i
    }
}

//构造覆盖
int covered=0; //被覆盖的顶点
MaxBin=SizeofB; //可能是非空的最大的链表
m=0; //C 的游标
while (MaxBin>0) { //搜索所有链表, 选择一个顶点
    if (bin[MaxBin]) { //链表不空
        int v=bin[MaxBin]; //第一个顶点
        C[m++]=v; //把 v 加入覆盖
        int j=Begin(v), k; //标记新覆盖的顶点
        while (j) {
            if (!Cov[j]) { //j 尚未覆盖
                Cov[j]=true;
                covered++;
                k=Begin(j); //修改 New
                while (k) {
                    New[k]--; //j 不计入在内
                    if (!Change[k]) {

```

```

        S.Add(k); //仅入栈一次
        Change[k]=true;
    }
    k=NextVertex(j);
}
}
j=NextVertex(v);
}
while (! S.IsEmpty()) { //更新链表
    S.Delete(k); Change[k]=false;
    MoveBins(SizeofB, New[k], k);
}
else MaxBub--;
}
delete [ ]New; delete [ ]Change; delete [ ]Cov;
return (covered==SizeofB);
}

```

2.2.4 回溯法

回溯法有“通用解题法”之称,可以用于系统地搜索问题的所有解。回溯法是一个既带有系统性又带有跳跃性的搜索算法。它在问题的解空间树中,按深度优先策略,从根结点出发搜索解空间树。算法搜索到解空间树的任意结点时,首先判断该结点是否包含问题的解。如果肯定不包含,则跳过对以该结点为根的子树的搜索,逐层向其祖先结点回溯;否则进入这棵子树,继续按深度优先策略搜索。用回溯法求解问题的所有解时,要回溯到根,而且根结点的所有子树都被搜索才结束。回溯法求解问题的一个解时,只要搜索到问题的一个解,就可以结束。这种系统搜索问题解的算法称为回溯法,它适用于求解组合数较大的问题。

用回溯法解问题时,应明确定义问题的解空间。问题的解空间至少应包含问题的一个(最优)解。此外,还应将解空间很好地组织起来,使得能用回溯法方便地搜索整个解空间。通常将解空间组织成树或图的形式。

确定了解空间的组织结构后,回溯法从开始结点(根结点)出发,以深度优先方式搜索整个解空间。这个开始结点成为活结点。在当前扩展结点处,搜索向纵深方向移至一个新结点。这个新结点成为新的活结点,并成为当前扩展结点。如果在当前扩展结点处不能再向纵深方向移动,则当前扩展结点成为当前死结点。此时,应该往回移动(回溯)至最近的活结点处,并使这个活结点成为当前扩展结点。回溯法以这种工作方式递归地在解

空间中搜索,直至找到所要求的解或解空间中已无活结点时为止。

回溯法搜索解空间树时,通常采用两种策略避免无效搜索,以提高回溯法的搜索效率。其一是使用约束函数,在扩展结点处剪去不满足约束的子树;其二是使用限界函数剪去得不到最优解的子树。这两种函数统称为剪枝函数。

用回溯法解决问题通常包含以下 3 个步骤:

- ① 针对所给问题,定义问题的解空间;
- ② 确定易于搜索的解空间结构;
- ③ 以深度优先方式搜索解空间,并在搜索过程中用剪枝函数避免无效搜索。

回溯法对解空间进行深度优先搜索,因此,在一般情况下可用递归方法实现:

```
void BackTrack(int t) {  
    //t 表示递归深度, n 用来控制递归深度, t > n 表示算法搜索到叶结点  
    if (t > n) output(x); //output(x) 记录或输出得到的可行解 x  
    else //f(n, t) 和 g(n, t) 分别表示当前扩展结点处还未搜索过的子树的起始和终止编号  
        for (int i = f(n, t); i <= g(n, t); i++) {  
            x[t] = h(i); //h(i) 表示在当前扩展结点处 x[t] 的第 i 个可选值  
            if (constraint(t) && bound(t)) BackTrack(t+1);  
        } //constraint(t) 和 bound(t) 是当前扩展结点处的约束函数和限界函数  
}
```

【例 2.4】 收费公路重建问题 设给定 n 个点 p_1, p_2, \dots, p_n 位于 x 轴上, 其中第一个点位于 0 处, x_i 是 p_i 的坐标。这 n 个点确定了在每一对点间的 $n(n-1)/2$ 个形如 $|x_i - x_j|$ 的距离。收费公路重建问题就是根据这些距离重新构建一个点的集合来安排这些点。解决收费公路重建问题的回溯法的算法如下:

```
int Turnpike(int x[], DistSet D, int n)  
{ //接收点的数组 x, 距离数组 D 和 n。如果找到一个解, 则返回 True。答案将放在数组 x  
  //中, 而 D 将是空集。否则, 返回 False。  
  x[1] = 0;  
  x[n] = DeleteMax(D); //删除点集中的最大者  
  x[n-1] = DeleteMax(D);  
  if (|x[n] - x[n-1]| ∈ D) {  
    Remove(x[n] - x[n-1], D); //从点集 D 中删除  
    Return Place(x, D, n, 2, n-2); //调用 Place 函数  
  }  
  else return False;  
}  
  
int Place(int x[], DistSet D, int n, int left, int right) {
```

```

int Dmax, Found=False;
if (empty(D)) return True;
Dmax=FindMax(D);
if ( |x[j]-Dmax| ∈ D for all 1<=j<left && right<j<=n) {
    x[right]=Dmax;
    for (1<=j<left && right<j<=n) Delete( |x[j]-Dmax|, D);
    Found=Place( x, D, n, left, right-1);
    if (! Found)
        for (1<=j<left && right<j<=n) Insert( |x[j]-Dmax|, D);
}

if (! Found && ( |x[n]-Dmax-x[j]| ∈ D for all 1<=j<left && right<j<=n) {
    x[left]=x[n]-Dmax;
    for (1<=j<left && right<j<=n)
        Delete( |x[n]-Dmax-x[j]|, D);
    Found=Place( x, D, n, left+1, right);
    if (! Found) //回溯
        for (1<=j<left && right<j<=n) Insert( |x[n]-Dmax-x[j]|, D);
}
return Found;
}

```

2.2.5 分支限界法

分支限界法类似于回溯法,是在问题的解空间上搜索问题解的算法,但求解的目标不同。回溯法的求解目标是找出解空间树中满足约束条件的所有解,而分支限界法的求解目标是找出满足约束条件的一个解,或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解,即在某种意义下的最优解。由于求解目标不同,所以两种方法对解空间树的搜索方式也不同。回溯法以深度优先方式搜索解空间树,而分支限界法则以广度优先或以最小耗费优先方式搜索解空间树。

问题的解空间树是表示问题空间的一棵有序树,常见的有子集树和排列树。在搜索问题的解空间树时,分支限界法与回溯法的主要不同在于它们对当前扩展结点所采用的扩展方式。在分支限界法中,每一个结点只有一次机会成为扩展结点。活结点一旦成为扩展结点,就一次产生其所有儿子结点。在这些儿子结点中,导致无可行解或导致非最优解的儿子结点被舍弃,其余儿子结点被加入活结点表中。此后,从活结点的表中取出下一个结点成为当前扩展结点,并重复上述结点扩展过程。这个过程一直持续到找到所需的

解或活结点的表为空时为止。

从活结点的表中选择下一个扩展结点的不同方式导致不同的分支限界法。最常见的有以下两种方式。

1) 队列式分支限界法

队列式分支限界法是将活结点表组织成一个队列,并按队列的先进后出原则选择一个结点为当前扩展结点。队列式分支限界法搜索解空间树的方式与解空间树的广度优先遍历算法极为相似。惟一的不同之处是队列式分支限界法不搜索已无可行结点为根的子树。

2) 优先队列式分支限界法

优先队列式分支限界法将活结点组织成一个优先队列,并按优先队列中规定的结点优先级选取优先级最高的下一个结点成为当前扩展结点。其中优先级常用一个与该结点相关的数值表示。在实现算法时,通常用最大堆来实现最大优先队列,以体现最大效益优先的原则。类似地,通常用最小堆来实现最小优先队列,以体现最小费用优先的原则。因此,在解决具体问题时,应根据问题的特点确定用最大优先队列或最小优先队列表示解空间的活结点的表。

在寻求问题的最优解时,与讨论回溯法相似,可以用剪枝函数加速搜索。

【例 2.5】 最大完备子图问题。设 U 是无向图 G 的顶点的子集,当且仅当对于 U 中任意的 u 和 v , (u, v) 是图 G 的一条边时, U 定义了一个完备子图。子图的尺寸为图中顶点的数量。当且仅当一个完备子图未包含在 G 的一个更大的完备子图中时,它是图 G 一个完备子图。最大的完备子图是最大尺寸的完备子图。解决最大完备子图问题的分支限界法的算法如下:

```
int AdjacencyGraph(int bestx[]) {  
    //寻找一个最大完备子图的最大收益分支限界算法  
    MaxHeap<CliqueNode> H(1000); //定义一个最多可容纳 1000 个活结点的最大堆  
  
    //初始化  
    bbnode * E=0; //当前的扩展结点为根  
    int i=1; //扩展结点的层  
    cn=0; //完备子图的大小  
    bestn=0; //目前最大完备子图的大小  
  
    //搜索子集空间树  
    while ( i != n+1) //不是叶子  
    { //在当前完备子图中检查顶点 i 是否与其他顶点相连
```

```

bool OK=true;
bbnode *B=E;
for (int j=i-1; j>0; B=B->parent, j--)
    if (B->Lchild && a[i][j]==NoEdge) {
        OK=false;        break;
    }
if (OK)    { //左孩子可行
    if (cn+1>bestn) best=n+1;
    AddCliqueNode(H, cn+1, cn+n-i+1, i+1, E, true);
}
if (cn+n-i>=bestn)    //右孩子有望
    AddCliqueNode(H, cn, cn+n-i, i+1, E, false);

//取下一个扩展结点
CliqueNode N;
H.DeleteMax(N); //不能为空
E=N.ptr;    Cn=N.cn;    i=N.level;

//沿着从扩展结点到根的路径构造 bestx[]
for (int j=n; j>0; j--) {
    bestx[j]=E->Lchild;
    E=E->parent;
}
return bestn;
}

```

2.2.6 随机算法

随机算法允许算法在执行过程中能随机地选择下一个计算步骤。在多数情况下,当算法在执行过程中面临一个选择时,随机性选择常常比最优选择省时。随机算法的一个基本特征是,对所求解问题的同一实例用同一随机算法求解两次可能得到完全不同的结果。这两次求解所需的时间以及结果可能会有很大差别。随机算法可分为四类:数值随机算法、蒙特卡洛(Monte Carlo)算法、拉斯维加斯(Las vegas)算法和舍伍德(Sherwood)算法。

在算法执行过程中,随机数至少有一次用于决策。随机算法的运行时间不仅依赖于特定的输入,而且依赖于所发生的随机数,即特定的输入不再是重要的,重要的是随机数。一个随机算法在最坏情形下的运行时间几乎总是与非随机算法在最坏情形下的运行时间

相同。对于一个好的随机算法来说,没有不好的输入,只有坏的随机数(相对于特定的输入)。

【例 2.6】 随机数发生器。随机数在随机算法的实现中起着十分重要的作用。实际上,计算机不可能产生真正的随机数,只能产生伪随机数。由于伪随机数具有随机数的大部分统计特性,所以在实践中,伪随机数可以满足要求。用于产生伪随机数的常用方法是线性同余法。

下面是线性同余法产生伪随机数的一种算法:

```
static unsigned long Seed=1; //unsigned long 型数据占据 32 个二进制位
#define A 48271L
#define M 2147483647L
#define Q (M/A)
#define R (M% A)

double Random(void) {
    long TmpSeed;
    TmpSeed=A * (Seed%Q) - R * (Seed/Q);
    if (TmpSeed>=0)    Seed=TmpSeed;
    else    Seed=TmpSeed+M;
    return (double)Seed/M;
}

void Initialize(unsigned long InitVal)
{    Seed=InitVal;
}
```

【例 2.7】 素性测试。关于素数的研究已有很长的历史,近代密码学的研究又给它注入了新的活力。下面介绍用费马小定理测试素数的方法。

费马小定理: 如果 p 是一个素数,且 $0 < a < p$, 则 $a^{p-1} \equiv 1 \pmod{p}$ 。

二次探测定理: 如果 p 是一个素数,且 $0 < x < p$, 则方程 $x^2 \equiv 1 \pmod{p}$ 的解为 $x=1$ 和 $x=p-1$ 。

在用费马小定理计算 $a^{n-1} \bmod n$ 的过程中,再利用二次探测定理增加对整数 n 的二次探测。一旦发现不满足二次探测的条件,即可得出 n 不是素数的结论。

```
HugeInt Witness(HugeInt A, HugeInt i, HugeInt N){
    HugeInt X,Y;
    if (i==0)    return 1;
```

```

X=Witness(A, i/2, N);
if (X==0) return 0;
Y=(X * X)%N;
if (Y==1 && X!=1 && X!=N-1) return 0;
if (i%2 !=0) Y=(A * Y)%N;
return Y;
}

int IsPrime(HugeInt N) { //判断大整数 N 是否素数
    return Witness(RandInt(2,N-2), N-1, N) == 1;
}

```

2.2.7 近似算法

迄今为止,所有的 NP 完全问题都还没有多项式时间算法。然而经常会遇到许多具有重要实际意义的 NP 完全问题。对于这类问题,通常可以采用以下几种解决策略。

① 只对问题的特殊实例求解。对一个 NP 完全问题,应仔细考虑是否必须实现一般意义的求解。也许只要针对某种特殊情形求解就够了,而针对特殊情形的求解常可得到高效的算法。

② 用动态规划法或分支限界法求解。这两种方法是求解许多 NP 完全问题的有效方法。

③ 用随机算法求解。若通过概率分析法证明某个 NP 完全问题的“难”实例是很少出现的。则可用随机算法求解,设计出高效的算法。

④ 只求近似解。由于问题的输入数据通常是用测量的方法得到的,因此输入的数据本身就是近似的。在实际中遇到的 NP 完全问题因此也不要求一定要获得非常精确的解答,只要求在一定误差范围内的近似解答就够了。许多解 NP 完全问题的近似算法可以用很少的时间获得很好的近似解。这是在实践中解决 NP 完全问题的非常有效且实用的方法。

⑤ 用启发式方法求解。这类方法通常是具体问题设计一些启发式搜索策略,以寻求问题的解,在实际使用时可能很有效。

许多 NP 完全问题实质上是最优化问题。若最优值为 c^* , 近似算法求得的近似最优解相应的 NP 完全问题数值为 c , 则将该近似算法的性能比定义为 $\eta = \max \left\{ \frac{c}{c^*}, \frac{c^*}{c} \right\}$ 。在通常情况下,定义的性能比是问题输入规模 n 的一个函数 $\rho(n)$, 表示为 $\max \left\{ \frac{c}{c^*}, \frac{c^*}{c} \right\} \leq$

$\rho(n)$ 。

有时用相对误差表示一个近似算法的精确程度会更方便些。近似算法的相对误差定义为 $\lambda = \left| \frac{c - c^*}{c^*} \right|$ 。若对于问题输入规模 n , 有一个函数 $\epsilon(n)$, 使得 $\left| \frac{c - c^*}{c^*} \right| \leq \epsilon(n)$, 则称 $\epsilon(n)$ 为该近似算法的相对误差界, 且有关系 $\epsilon(n) \leq \rho(n) - 1$ 。

许多问题的近似算法具有固定的性能比和相对误差界, 即 $\rho(n)$ 或 $\epsilon(n)$ 是不随 n 的变化而变化的。有些 NP 完全问题可以找到这样的近似算法, 其性能比可以通过增加计算量来改进。也就是说, 在计算量和解的精确度之间有一个折中。

【例 2.8】集合覆盖问题。集合覆盖问题指的是一个实例 $\langle X, F \rangle$ 由一个有限集 X 及 X 的一个子集族 F 组成。子集族 F 覆盖了有限集 X 。也就是说, X 中每一个元素至少属于 F 中的一个子集, 即 $X = \bigcup_{S \in F} S$ 。对于 F 中的一个子集 $C \subseteq F$, 若 C 中的 X 的子集覆盖了 X , 即 $X = \bigcup_{S \in C} S$, 则称 C 覆盖了 X 。集合覆盖问题就是要找出 F 中覆盖 X 的最小子集 C^* , 使得

$$|C^*| = \min\{|C| \mid C \subseteq F \text{ 且 } C \text{ 覆盖 } X\}$$

对于集合覆盖问题, 可以设计出一个简单的贪心算法, 求该问题的一个近似最优解。这个近似算法具有对数性能比。算法描述如下:

```
Set GreedySetCover( $X, F$ ) {
     $U = X$ ;
     $C = \emptyset$ ;
    while ( $U \neq \emptyset$ ) {
        选择  $F$  中使  $|S \cap U|$  最大的子集  $S$ ;
         $U = U - S$ ;
         $C = C \cup \{S\}$ ;
    }
    return  $C$ ;
}
```

第3章 操作系统知识

3.1 操作系统基础知识

计算机系统软件通常分为系统软件和应用软件两大类。应用软件是指计算机用户利用计算机的软件、硬件资源为某一专门的应用目的而开发的软件。例如科学计算、工程设计、数据处理、事务处理和过程控制等方面的程序,以及文字处理软件、表格处理软件、辅助设计软件(CAD)和实时处理软件等。系统软件是计算机系统的一部分,由它支持应用软件的运行,为用户开发应用系统提供一个平台。用户可以使用它,但不能随意修改它。常用的系统软件有操作系统、语言处理程序、连接程序、诊断程序以及数据库管理系统等。操作系统是计算机系统中的核心软件,其他软件均建立在操作系统的基础上,并在操作系统的统一管理和支持下运行。

3.1.1 操作系统的定义与作用

计算机系统的硬件资源包括中央处理机(CPU)、存储器(包括主存与外存)和输入输出设备等物理设备。计算机系统的软件资源是以文件形式保存在存储器上的程序和数据等信息。操作系统能有效地组织和管理系统中的各种软、硬件资源,合理地组织计算机系统的工作流程,控制程序的执行,并且向用户提供一个良好的工作环境和友好的接口。操作系统有两个重要的作用。

1. 通过资源管理,提高计算机系统的效率

操作系统是计算机系统的资源管理者,它含有对系统软、硬件资源实施管理的一组程序。其首要作用就是通过CPU管理、存储管理、设备管理和文件管理,对各种资源进行合理的分配,改善资源的共享和利用程度,最大限度地发挥计算机系统的工作效率,提高计算机系统在单位时间内处理工作的能力(即系统的“吞吐量”throughput)。

2. 改善人机界面,向用户提供友好的工作环境

操作系统不仅是计算机硬件和各种软件之间的接口,也是用户与计算机之间的接口。试想如果不安装操作系统,用户将要面对的是由0、1组成的代码和一些难懂的机器指令,通过按钮或按键来操作计算机,这样既笨拙,又费时。一旦安装操作系统后,用户面对的不再是笨拙的裸机,而是操作便利、服务周到的操作系统,明显改善用户界面,提高了用

户的工作效率。

3.1.2 操作系统的特征与功能

1. 操作系统的特征

操作系统的 4 个特征是并发运行、共享性、虚拟性和不确定性。

(1) 并发性(concurrency): 指在计算机系统中存在着许多并发运行的活动。对计算机系统而言,并发是指从宏观上看系统内有多道程序同时运行,微观上看是串行运行。因为大多数计算机系统中一般只有一个 CPU,在任意时刻只能有一道程序占用 CPU。

(2) 共享性(sharing): 系统中各个并发活动要共享计算机系统中的各种软、硬件资源,因此操作系统必须解决在多道程序间合理地分配和使用资源的问题。

(3) 虚拟性(virtual): 虚拟性是操作系统中的重要特征。所谓虚拟性是指把物理上的一台设备变成逻辑上的多台设备。例如,在操作系统中采用了假脱机(spooling)技术,可以利用快速、大容量可共享的磁盘作为中介,模拟多个非共享的低速输入输出设备,这样的设备称为虚拟设备。

(4) 不确定性(non-determinacy): 通常一个程序的初始条件相同时,无论何时运行结果必然相同。但由于程序的并发运行,系统内的各种进程错综复杂,与这些进程有关的事件(如从外部设备来的中断、输入输出请求和各种运行故障等)发生的时间都不可预测,如果处理不当,将导致系统出错,这种不确定性所带来错误是很难查找的。

2. 操作系统的功能

操作系统的 5 大管理功能如下所述。

(1) 进程管理: 实质上是对处理机执行“时间”的管理,采用多道程序等技术将 CPU 真正合理地分配给每个任务。

包括进程控制(创建、撤销、挂起和改变运行优先级等)、进程同步(协调并发进程之间的推进步骤,以协调资源共享)、进程通信(进程之间传送数据,以协调进程间的协作)和进程调度(作业和进程的运行切换,以充分利用处理机资源和提高系统性能)。

(2) 文件管理: 又称为信息管理。

包括文件存储空间管理(解决如何存放信息,以提高空间利用率和读写性能)、目录管理(解决信息检索问题)、文件的读写管理和存取控制(解决信息安全问题,可通过系统设置用户口令,对用户分类,设置文件权限)和软件管理(软件的版本和相互依赖关系,安装和卸载等)。

(3) 存储管理: 是对主存储器“空间”进行管理。

包括存储分配与回收(操作系统为每个程序分配存储空间,当程序运行完毕回收存储空间)、存储保护(保证进程间互不干扰,相互保密,如访问合法性检查,防止从“垃圾”中窃

取其他进程的信息)地址映射或变换(进程逻辑地址到主存物理地址的映射)以及主存扩充(包括覆盖、交换和虚拟存储。用于提高主存利用率,扩大进程的主存空间)。

(4) 设备管理: 实质是对硬件设备的管理, 其中包括对输入输出设备的分配、启动、完成和回收。操作系统中含有许多设备驱动程序, 用户和应用程序使用外部设备时并不需要知道外部设备的具体特性, 不需要对设备的使用专门编程, 对设备的具体操作都由设备驱动程序完成。这样不但简化了程序设计, 而且程序运行不依赖于具体硬件配置, 做到了与“硬件无关”。

(5) 作业管理: 包括任务、界面管理、人机交互、图形界面、语音控制和虚拟现实等。

操作系统提供系统命令一级的接口, 供用户控制自己的作业运行, 如命令行、菜单式 GUI“联机”或命令脚本“脱机”。操作系统还提供编程一级接口, 供用户程序和系统程序调用操作系统功能, 如系统调用和高级语言库函数。

3.1.3 操作系统的类型

操作系统分为批量处理操作系统(简称批处理)、分时操作系统、实时操作系统、网络操作系统、分布式操作系统、微机操作系统和嵌入式操作系统。

1. 批处理操作系统

在早期的计算机系统中, 程序的每一次运行都需要人工干预, 操作过程繁琐, 占用很多人工等待的时间, 也很容易产生错误, 可真正执行程序的时间却很短。而且程序在执行的过程中, 要独占系统的全部硬件资源, 利用率很低, 为此引入了批处理操作系统。批处理操作系统分为单道批处理和多道批处理两种。

在批处理系统中, 作业由用户程序、数据和作业说明书(作业控制语言)3 部分组成。

批处理系统的优点: 同一批作业内各作业。自动依次执行, 改善了主机 CPU 和 I/O 设备的使用效率, 提高了吞吐量。

批处理系统的缺点: 磁带或磁盘需要人工装卸, 作业需要人工分类, 监督程序易遭到用户程序的破坏(由人工干预才可恢复)。其次, 由于一次要处理一批作业, 在该批作业的处理过程中, 任何用户都不能与计算机进行交互。即使发现了某个作业的程序有错, 也要等该批作业全部结束后才能修改。这对于软件开发人员来说, 不能不说是严重的缺陷。正是这一问题, 导致了分时操作系统应运而生。

2. 分时系统

分时操作系统将 CPU 的工作时间划分为许多很短的时间片, 轮流为各个终端的用户服务。分时系统的特点如下所述。

(1) 多路性: 允许在一台主机上同时连接多台联机终端, 系统按分时原则为每个用户服务。宏观上是多个用户同时工作, 共享系统资源, 而微观上则是每个用户作业轮

流运行一个时间片。多路性即同时性,它提高了资源利用率,从而促进了计算机的广泛应用。

(2) 独立性: 每个用户各占一个终端,彼此独立操作,互不干扰。因此用户会感觉到就像自己一人独占主机。

(3) 交互性: 用户可通过终端与系统进行人机对话。用户可以请求系统提供多方面服务,如文件编辑、数据处理和资源共享等。

(4) 及时性: 用户的请求能在很短时间获得响应,此时间间隔是以人们所能接受的等待时间确定的,通常为 $1\sim 2s$ 。响应时间是分时系统的重要指标,它是从用户发出终端命令到系统做出响应间的时间间隔。系统的响应时间主要是根据用户所能接受的等待时间确定的。分时系统中时间片的选择是一个复杂和关键的任务,如时间片选得过大,响应时间不变时导致用户数减少,或造成响应时间过长。当时间片过小时,时间片切换的开销相对增加,一个进程相对要花费更多的时间片才能结束,进程在系统中的周转时间大大加长。最佳的时间片值应既能使分时用户得到好的响应时间,同时又要使时间片切换的开销相对较小因而可以忽略。UNIX 系统是典型的多用户、多任务的分时操作系统。

3. 实时系统

实时是指计算机对于外来信息能够以足够快的速度进行处理,并在被控对象允许的时间范围内做出快速反应。实时系统对交互能力要求不高,但要求可靠性有保障。为了提高系统的响应时间,对随机发生的外部事件应及时做出响应并进行处理。

实时系统可分为实时控制系统和实时信息处理系统。实时控制系统主要用于生产过程的自动控制,实验数据的自动采集,武器的控制(包括火炮自动控制、飞机自动驾驶和导弹制导系统)。实时信息处理系统主要用于实时信息处理,如飞机订票系统和情报检索系统。实时系统的主要特点如下。

(1) 快速的响应时间: 实时系统是为了提高系统响应时间而设计的操作系统,特别是实时控制系统,对外部事件的响应要十分及时迅速。外部事件往往以中断方式通知系统,系统应有较强的中断处理能力。实时系统的设计也以“事件驱动”方式来设计。

(2) 有限的交互能力: 实时系统(如实时信息处理系统)一般是专用系统,它能提供人机交互方式,但用户只能访问系统中某些特定的专用服务程序,不能像分时系统那样为终端用户提供多方面服务。

(3) 高可靠性: 批处理系统和分时系统虽也要求系统可靠,相比之下,实时系统则要求系统高度可靠。因此实时系统中往往都采用双机系统和多级容错措施来保证系统和数据的安全。

实时系统与分时系统除了应用的环境不同,主要区分如下所述。

(1) 系统的设计目标不同：分时系统是一个设计成多用户的通用系统，而实时系统大都是专用系统。

(2) 交互性的强弱：分时系统是多用户的通用系统，交互能力强。而实时系统是专用系统，仅允许访问有限的专用程序，且交互能力差。

(3) 响应时间的敏感程度不同：分时系统是以人能接收的等待时间为系统的设计依据，而实时系统是以被测物体所能接受的延迟为系统设计依据，因此实时系统对响应时间的敏感程度强，而分时系统的敏感程度弱。

4. 网络操作系统

网络操作系统是使联网地计算机能方便而有效地共享网络资源，为网络用户提供所需各种服务的软件和有关协议的集合。因此网络操作系统的功能主要包括高效、可靠的网络通信，对网络中共享资源（在 LAN 中有硬盘、打印机等）的有效管理，提供电子邮件、文件传输、共享硬盘及打印机等服务，网络安全管理，提供互操作能力。

5. 分布式操作系统

分布式计算机系统是由多个分散的计算机经网络连接而成的计算机系统，系统中的计算机无主、次之分，任意两台计算机都可以通过通信交换信息。为分布式计算机配置的操作系统称“分布式操作系统”。

分布式操作系统能直接对系统中各类资源进行动态的分配和调度，任务划分、信息传输协调工作，并为用户提供一个统一的界面和标准接口，用户通过这一界面实现所需要的操作和使用系统资源，使系统中若干台计算机相互协作完成共同的任务，有效地控制和协调诸任务的并行执行。

分布式操作系统是网络操作系统的更高级形式，它保持网络系统所拥有的全部功能，同时又有透明性、可靠性和高性能等。网络操作系统与分布式操作系统虽然都用于管理分布在不同地理位置的计算机，但最大的差别是：网络操作系统的工作时必须确认网址，而分布式系统则不必知道计算机的确切地址；分布操作系统负责整个系统的资源分配，通常能够很好地隐藏系统内部的实现细节，如对象的物理位置和并发控制等。这些对用户都是透明的。

6. 微机操作系统

微型计算机拥有巨大的使用量和广泛的用户。配置在微型计算机上的操作系统称为微机操作系统。常用的微机操作系统有 MS-DOS、MS Windows、OS/2、SCO UNIX 和 Linux 等。其中，Microsoft 公司开发的单用户单任务操作系统 MS-DOS 是首先在 IBM-PC 机上使用的微机操作系统，MS-DOS 操作系统现在成了事实上的 16 位微机单用户单任务操作系统的标准。

多任务操作系统 Windows 98/NT/2000/XP 是 Microsoft 公司开发的一系列图形用

户界面的多任务、多线程的操作系统。SCO UNIX 是 SCO 公司将运行于大、中、小型机上 UNIX 操作系统移植到微机上的多用户多任务操作系统。

Linux 操作系统是一个遵循标准操作系统界面的免费操作系统,具有 UNIX BSD 和 UNIX system V 的扩展特性。它的创造者是芬兰籍的 Linus B. Torvalds 和其他开发人员,并且遵循通用公共许可证 (General Public License, GPL),保证用户有使用上的自由,获得源程序的自由,修改的自由,以及可以复制和推广的自由。

7. 嵌入式操作系统

嵌入式操作系统是运行在嵌入式智能芯片环境中,对整个智能芯片及其控制的各种部件和装置等资源进行统一协调、处理、指挥和控制的系统软件。

3.1.4 研究操作系统的观点

1. 资源管理的观点

从资源管理的观点来看,操作系统管理的对象是计算机系统的资源,操作系统则是管理计算机系统的程序集合。这种观点是在共享的前提下以资源分配、使用和回收为出发点,考虑操作系统各部分程序的功能和算法,解决并发环境中的资源管理问题。通常将操作系统分为处理机管理、存储管理、设备管理、文件管理以及用户与操作系统接口等 5 个主要部分。主要研究如下的问题。

(1) 记住资源的使用情况。如哪些资源是空闲的,哪些资源正在使用,被谁使用等。

(2) 确定资源的分配策略。根据各类资源的不同特点确定一组策略,以决定资源的分配和调度。

(3) 分配/回收资源。根据用户的要求和资源的分配策略分配资源,当用户作业不再需要某种资源时,及时回收资源,以便重新分配给其他用户。

2. 虚拟机的观点

按照虚拟机 (virtual machine) 的观点,操作系统加裸机 (bare machine) 等于虚拟计算机。即在一台纯粹由硬件组成的裸机基础上安装操作系统后,将变成一台比裸机功能更强的、使用更加方便的“虚拟”计算机。操作系统的全部功能,如系统调用、命令和作业控制语言等,称为操作系统虚拟机。该虚拟机向用户提供全套的服务,完成用户要求。

事实上,虚拟机的观点是从功能分解的角度出发,考虑操作系统的结构,将操作系统分成若干层次,每一层完成特定功能,并为上一层提供支持,构成它的运行环境。这样,通过逐个层次的功能扩充最终完成操作系统虚拟机,从而向用户提供各种服务,完成用户的各项任务。

3.2 处理机管理

在多道程序批处理系统和分时系统中,有多个并发执行的程序。此时用程序这个静态的概念已经不能描述系统中程序动态变化的过程,所以引入了进程概念。进程是资源分配和独立运行的基本单位。研究操作系统的进程,实质上是研究系统中诸进程之间的并发特性以及进程之间的相互制约性。

3.2.1 基本概念

1. 程序与进程

1) 程序的顺序执行

一个较大的程序通常都由若干个程序段组成。程序在执行时,各程序段必须按照先后次序逐个执行。程序各程序段先后执行的次序关系可用前趋图表示。

前趋图是一个有向无循环图,图由结点和结点间的有向边组成。结点代表各程序段的操作,而结点间的有向边表示两程序段的操作之间存在的前趋关系(“ \rightarrow ”)。两程序段 P_i 和 P_j 的前趋关系表示成 $P_i \rightarrow P_j$, P_i 是 P_j 的前趋, P_j 是 P_i 的后继。图 3-1 为 3 个程序段,第 1 个程序段“输入”是第 2 个程序段“计算”的前驱,第 2 个程序段“计算”是第 3 个程序段“输出”的前驱。

程序顺序执行时的特征如下所述。

- 顺序性: 程序各程序段严格按照规定的顺序执行。
- 封闭性: 程序运行时系统内各资源只受该程序控制,执行结果不受外界因素影响。
- 可再现性: 只要程序执行环境和初始条件相同,不管程序执行多少次,可获得相同的结果。



图 3-1 3 个结点的前趋图

2) 程序并发执行的特征

若在计算机系统中采用多道程序设计技术,则主存中的多道程序可处于并发执行状态。对于上述有 3 个程序段的作业,虽然每个作业有前趋关系的各程序段不能在 CPU 和输入输出各部件并行执行,但同一个作业内没有前趋关系的程序段或不同作业的程序段可以分别在 CPU 和各输入输出部件上并行执行。例如,某系统中有一个 CPU、一台输入设备和一台输出设备。每个作业具有 3 个程序段:输入 I_i , 计算 C_i 和输出 P_i ($i=1, 2, 3$)。图 3-2 为 3 个作业的各程序段并发执行的前趋图。

从图 3-2 中可以看出, I_2 与 C_1 并行执行, I_2 、 C_2 与 P_1 并行执行, C_3 与 P_2 并行执行。

其中 I_2 、 I_3 受到 I_1 的间接制约, C_2 、 C_3 受到 C_1 的间接制约, P_2 、 P_3 受到 P_1 的间接制约。同理, C_1 、 P_1 受到 I_1 的直接制约等。

程序并发执行时的特征如下:

- (1) 失去了程序的封闭性;
- (2) 程序和机器执行程序的活动不再一一对应;
- (3) 并发程序间的相互制约性。

例如,假定两个并发执行的程序段的功能是完成交通流量的统计。其中“观察者”P1 识别通过的车辆数,“报告者”P2 定时将观察者的计数值清“0”。程序实现如下:

P1

L1: if 有车通过 then
COUNT := COUNT + 1;
GOTO L1;

P2

L2: PRINT COUNT;
COUNT := 0;
GOTO L2;

对于上例,由于程序可并发执行,所以可能有以下 3 种执行序列:

- ① COUNT := COUNT + 1; PRINT COUNT; COUNT := 0
- ② PRINT COUNT; COUNT := 0; COUNT := COUNT + 1
- ③ PRINT COUNT; COUNT := COUNT + 1; COUNT := 0

假定 COUNT 在某个循环的初值为 n ,那么对这 3 种执行序列得到的 COUNT 结果是不同的,如表 3-1 所示。

表 3-1 程序并发执行的结果

执行序列	①	②	③
COUNT 打印的值	$n + 1$	n	N
COUNT 执行后的值	0	1	0

这种不正确的结果的发生是因为两个程序共享变量 COUNT 引起的,为了解决这一问题,需要研究进程间的同步与互斥问题。

引入进程的原因是由于程序并发执行破坏了程序的封闭性和可再现性,使得程序和执行程序的活动不再一一对应。此时用程序这个静态的概念已经不能描述系统中程序动态执行的过程,所以引入了进程。

2. 进程的组成

进程是程序的一次执行,该程序可以和其他程序并发执行。进程通常是由程序、数据

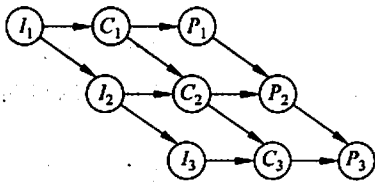


图 3-2 程序并发执行的前趋图

及进程控制块(PCB)组成的。

(1) 进程的**程序部分**描述了进程需要完成的功能。假如一个程序能同时被多个进程共享,那么这一部分就应该以可再入(纯)码的形式编制,它是程序执行时不可修改的部分。

(2) 进程的**数据集合部分**包括程序执行时所需的数据及工作区。这部分只能为一个进程所专用,是进程的可修改部分。

(3) **进程控制块**是进程的描述信息和控制信息,是进程动态特性的集中反映,也是进程存在的惟一标志。进程控制块主要包含的内容如表 3-2 所示。

表 3-2 PCB 的内容

信 息	含 义
进程标识符	标明系统中的各个进程
状态	说明进程当前的状态
位置信息	指明程序及数据在主存或外存的物理位置
控制信息	参数、信号量、消息等
队列指针	链接同一状态的进程
优先级	进程调度的依据
现场保护区	将处理机的现场保护到该区域以便再次调度时能继续正确运行
其他	因不同的系统而异

3. 进程的状态及其状态间的切换

1) 三态模型

在多道程序系统中,进程在处理器上交替运行,状态也不断地发生变化,因此进程一般有运行、就绪和阻塞 3 种基本状态。图 3-3 显示了进程基本状态及其转换,也称三态模型。

(1) **运行**: 当一个进程在处理器上运行时,则称该进程处于运行状态。显然对于单处理机系统,处于运行状态的进程只有一个。

(2) **就绪**: 一个进程获得了除处理机外的一切所需资源,一旦得到处理机即可运行,则称此进程处于就绪状态。

(3) **阻塞**: 也称等待或睡眠状态。一个进程正在等待某一事件发生(如等待 I/O 完成等)而暂时停止运行,这时即使把处理机分配给进程也无法运行,故称该进程处于阻塞状态。

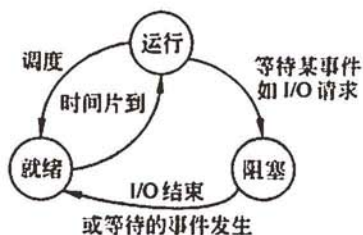


图 3-3 进程的三态模型

2) 五态模型

事实上,对于一个实际的系统,进程的状态及其转换将更复杂。例如,引入新建态和终止态构成了进程的五态模型。如图 3-4 所示。

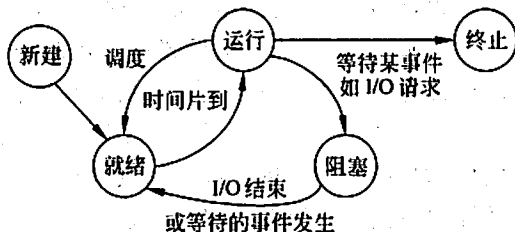


图 3-4 进程的五态模型

其中,新建态对应于进程刚刚被创建时还没有被提交,并等待系统完成创建进程的所有必要信息的状态。因为创建进程时分为两个阶段,第一个阶段是为一个新进程创建必要的管理信息,第二个阶段让该进程进入就绪状态。由于有了新建态,操作系统往往可以根据系统的性能和主存容量的限制推迟新建态进程的提交。

类似地,进程的终止也可分为两个阶段:等待操作系统进行善后处理;释放主存。

3) 具有挂起状态的进程状态及其转换

由于进程的不断创建,系统资源特别是主存资源已不能满足所有进程的运行要求。这时,就必须将某些进程挂起,放到磁盘对换区,暂时不参加调度,以平衡系统负载。进程挂起的原因可能是系统出现故障,用户调试程序,也可能是需要检查问题。图 3-5 是具有挂起状态的进程状态及其转换。

活跃就绪是指进程已在主存并且可以被调度的状态。

静止就绪是指进程被对换到辅存时的就绪状态,不能被直接调度。只有当主存中没有活跃就绪态进程,或者是静止就绪进程具有更高的优先级,系统将把静止就绪态进程调回主存并转换为活跃就绪。

活跃阻塞是指进程已在主存,一旦等待的事件产生便进入活跃就绪状态。

静止阻塞是指进程对换到辅存时的阻塞状态,一旦等待的事件产生便进入静止就绪状态。

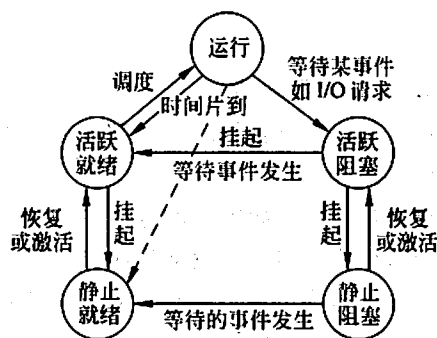


图 3-5 细分进程状态及其转换

3.2.2 进程的控制

进程的控制就是对系统中所有进程从创建到消亡的全过程实施有效的控制。这意味着不仅要控制正在运行的进程,而且还要能创建新的进程,撤销已完成的进程。

为了对进程进行有效的控制,操作系统必须设置一套控制机构。进程的控制机构是由操作系统内核实现的。通常将与硬件密切相关的模块放在紧挨硬件的软件层中,并使它们常驻主存,以便提高操作系统的运行效率。例如,中断处理程序、常用的设备驱动程序以及运行频率较高的模块(如时钟中断、进程调度和其他许多公用的基本模块)。通常将这一部分称之为操作系统的内核。它为系统实现进程控制和存储器管理,提供了有效的控制机制。不同的操作系统内核所包含的功能不同,但大多数操作系统的内核均包含支撑功能和资源管理的功能。

1. 支撑功能

(1) 中断处理:操作系统的各种重要活动最终都依赖于中断。如各种类型的系统调用、键盘命令的输入、设备驱动及文件系统等都依赖于中断。通常内核只对中断进行“有限次处理”,然后转入有关进程继续处理。这不仅可以减少中断处理的时间,也可以提高程序的并发性。

(2) 时钟管理:操作系统的许多活动要用到它。如在分时系统时间片调度算法中,当时间片用完时,由时钟管理产生一个中断信号,通知调度程序重新调度。其他如实时系统中的截止时间控制,批处理系统中的最长运行时间控制等。

(3) 原语操作:内核在执行某些基本操作时,往往是通过原语操作实现的。原语是由若干条机器指令构成的,用于完成特定功能的一段程序。原语在执行的过程中是不可分割的。

2. 资源管理功能

进程管理:进程管理的大部分或全部功能都放在内核中,如进程的调度与分配、进程的创建和撤销。

存储器管理:如内存的分配和回收、内存的保护和对换等功能模块都放在内核中。

设备管理:如设备驱动程序、缓冲区管理和设备分配等功能模块都放在内核中。

内核中包含的原语主要有:进程控制、进程通信、资源管理以及其他方面的原语。

3. 进程控制原语

进程控制原语主要有:进程创建原语、进程撤销原语、进程挂起原语、进程激活原语、进程阻塞原语以及进程唤醒原语。

1) 创建原语与撤销原语

(1) 创建原语:创立一个进程实际上就是先为其创建一个PCB,设置好PCB的各项

参数,其次装入该进程的实体(程序和数据)。通常有两种情况需要创建进程:在系统生成时建立的一些系统进程,如系统的输入读进程和输出写进程;通过创建原语产生进程。通常,创建原语创建的进程主要是非常驻主存的系统进程和用户进程。

(2) 撤销原语:一旦操作系统发现了要求终止进程的事件后,便调用进程撤销原语终止一个指定的进程。撤销原语将读取进程的状态。若被终止的进程处于“运行”状态,应立即中断其执行,保护 CPU 现场,设置重新调度标志;若有子孙进程,还应终止所有的子孙进程,以免其成为不可控的;收回所有的资源,或者归还父进程,或者归还系统;将被终止的进程从队列中移出。

引起进程终止的事件有正常结束、异常结束(如越界错误、保护错、特权指令错、运行超时、等待超时及 I/O 故障)、外界干预(操作员或操作系统干预、父进程请求及父进程终止)等。

2) 挂起原语与激活原语

(1) 挂起原语:当出现了引起进程挂起的事件时,挂起原语将检查被挂起进程的状态。调用挂起原语的进程只能挂起它自己或它的子孙,而不能挂起别的族系的进程。挂起原语的执行过程是:检查要挂起进程 PCB 的现行状态,若正处于活动就绪态,便将它改为静止就绪态,如是活动阻塞态则改为静止阻塞态。如是运行态,则将它改为静止就绪态,并调用进程调度程序重新分配处理机。为了方便用户或父进程考察该进程的运行情况,需把该进程的 PCB 复制到主存指定区域。

(2) 激活原语:当发生了激活进程的事件时,如用户进程或父进程请求激活指定的进程时,激活原语将检查被激活进程的状态。若进程处于静止就绪,则置为活动就绪;若进程处于静止阻塞,则置为活动阻塞。

3) 阻塞原语与唤醒原语

阻塞原语与唤醒原语:当正在执行的进程请求操作系统提供服务时,由于请求的设备或资源得不到满足,此时只有用阻塞原语将其阻塞起来,一旦等待的资源空闲,则用唤醒原语唤醒该进程。引起进程阻塞的主要原因如下所述。

- 启动某种操作:如果进程启动某操作后,必须等待操作结束才能继续运行,如启动 I/O 设备,当 I/O 操作结束了,由中断处理进程将其唤醒。
- 新数据尚未到达:当两个相互合作的进程,一个进程必须等待另一个进程的结果时,必须阻塞一个进程,当等待的事件发生了,再将其唤醒。
- 无新工作可做:系统中往往设置一些具有特定功能的系统进程,每当任务完成后,这些进程将自己阻塞起来,当有新的任务再将其唤醒。

3.2.3 进程间的通信

1. 同步与互斥

在计算机系统中,多个进程可以并发执行,因此进程间必然存在共享资源和相互合作的问题。

1) 进程间的同步

一般来说,一个进程相对于另一个进程的运行速度是不确定的,也就是说进程是在异步环境下运行的。每个进程都以各自独立的、不可预知的速度向前推进。但相互合作的进程需要在某些确定点上协调它们的工作,当一个进程到达了这些点后,除非另一进程已经完成了某些操作,否则就不得不停下来等待这些操作结束。这就是进程间的同步。

2) 进程间的互斥

在多道程序系统中,各进程可以共享各类资源,但有些资源一次只能供一个进程使用,称为临界资源(critical resource, CR),如打印机、公共变量和表格等。

同步是进程间的直接制约问题,互斥是进程间的间接制约问题。

3) 临界区管理的原则

临界区(critical section, CS)是进程中对临界资源实施操作的那段程序。互斥临界区管理的原则如下所示。

- 有空即进: 无进程处于临界区时,允许进程进入临界区,并且只能在临界区运行有限的时间。
- 无空则等: 临界区中有进程时,其他欲进入临界区的进程必须等待,以保证进程互斥地访问临界资源。
- 有限等待: 对要求访问临界资源的进程,应保证进程能在有限时间进入临界区,以免陷入“饥饿”状态。
- 让权等待: 当进程不能进入自己的临界区时,应立即释放处理机。

2. 信号量机制

1965年,荷兰学者 Dijkstra 提出的信号量机制是一种卓有成效的进程同步与互斥的工具。目前信号量机制有了很大的发展,主要有整型信号量、记录型信号量以及信号量集机制。

1) 整型信号量与 PV 操作

信号量是一个整型变量,根据控制对象的不同赋不同的值。信号量分为两类。

- 公用信号量: 实现进程间的互斥,初值=1 或资源的数目;
- 私用信号量: 实现进程间的同步,初值=0 或某个正整数。

信号量 S 的物理意义 $S \geq 0$ 表示某资源的可用数, $S < 0$ 其绝对值表示阻塞队列中等

待该资源的进程数。

对系统中的每一个进程,其工作的正确与否不仅取决于它自身的正确性,而且与它在执行中能否与其他相关进程正确地实施同步与互斥有关。PV 操作是实现进程同步与互斥的常用方法。PV 操作是低级通信原语,在执行期间不可分割。其中,P 操作表示申请一个资源,V 操作表示释放一个资源。

P 操作定义: $S := S - 1$, 若 $S \geq 0$, 则执行 P 操作的进程继续执行; 否则, 若 $S < 0$, 则置该进程为阻塞状态(因为无可用资源), 并将其插入阻塞队列。

P 操作可用如下过程表示:

```
Procedure P(Var S; Semaphore);
Begin
  S := S - 1;
  If S < 0 then W(S)      {执行 P 操作的进程插入阻塞队列}
End;
```

V 操作定义: $S := S + 1$, 若 $S > 0$, 则执行 V 操作的进程继续执行; 否则, 若 $S \leq 0$, 则从阻塞状态唤醒一个进程, 并将其插入就绪队列, 执行 V 操作的进程继续执行。

V 操作可用如下过程表示:

```
Procedure V(Var S; Semaphore);
Begin
  S := S + 1;
  If S <= 0 then R(S)      {从阻塞队列中唤醒一个进程}
End;
```

2) 利用 PV 操作实现进程的互斥

令信号量 mutex 的初值为 1, 当进程进入临界区时执行 P 操作, 退出临界区时执行 V 操作。则进入临界区的代码段如下:

```
P(mutex)
临界区
V(mutex)
```

【例 3.1】 将交通流量统计程序改写如下:

P1	P2
L1: if 有车通过 then	L2: begin
begin	P(mutex)
P(mutex)	PRINT COUNT;
COUNT := COUNT + 1;	COUNT := 0;

```
V(mutex)
end
GOTO L1;
```

```
V(mutex)
end
GOTO L2;
```

3) 利用 PV 操作实现进程的同步

进程的同步是由于进程间合作而引起的相互制约的问题。要实现进程的同步,可用一个信号量与消息联系起来。当信号量的值为“0”时表示希望的消息未产生,当信号量的值为非“0”时表示希望的消息已经存在。假定用信号量 S 表示某条消息,进程可以通过调用 P 操作测试消息是否到达,调用 V 操作通知消息已准备好。最典型的是单缓冲区的生产者和消费者的同步问题。

【例 3.2】 生产者进程 $P1$ 不断地生产产品送入缓冲区,消费者进程 $P2$ 不断地从缓冲区中提取产品消费。为了实现 $P1$ 与 $P2$ 进程间的同步,需要设置一个信号量 $S1$,且初值为 1,表示缓冲区空,可以将产品送入缓冲区;设置另一个信号量 $S2$,且初值为 0,表示缓冲区有产品。其同步过程如图 3-6 所示。

若信号量 $S1$ 的初值可以设为 0,此时同步过程图 3-7 所示。



图 3-6 单缓冲区的同步举例方法 1



图 3-7 单缓冲区的同步举例方法 2

【例 3.3】 设有一个生产者和一个消费者,缓冲区可存放 n 件物品。生产者不断地生产产品,消费者不断地消费产品。如何用 PV 操作实现生产者和消费者的同步? 可以设置 3 个信号量 S 、 $S1$ 和 $S2$,其中, S 是一个互斥信号量且初值为 1,因为缓冲区是一个互斥资源,所以需要进行互斥控制; $S1$ 表示是否可以将物品放入缓冲区,初值为 n ; $S2$ 表示缓冲区是否存有物品,初值为 0。同步过程如图 3-8 所示。

3. 高级通信原语

进程间通信是指进程之间的信息交换。少则一种状态,多则成千上万个信息,若用前面介绍的 PV 操作实现则存在如下问题:

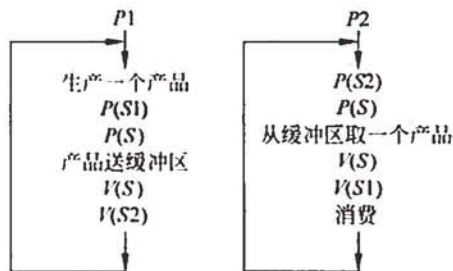


图 3-8 缓冲区容量为 n 件的同步举例

- 编程难度大,通信对用户不透明,即需要用户利用低级通信工具实现进程间的同步与互斥。若使用不当则易引起死锁。
- 效率低,生产者每次只能向缓冲区放一个消息,消费者只能从缓冲区中取一个消息。为此引入高级通信原语。进程高级通信的类型如下所示。

(1) 共享存储系统: 相互通信的进程共享某些数据结构或存储区,实现进程之间的通信。

(2) 消息传递系统: 进程间的数据交换以消息为单位,程序员可以直接利用系统提供的一组通信命令(原语)来实现通信。如 Send(A)和 Receive(A)。

(3) 管道通信: 管道通信是以文件系统为基础的。所谓管道是指用于连接两个进程之间的一个打开的共享文件(pipe 文件)。向管道(共享文件)提供输入的发送进程(即写进程)以字符流的形式将大量的数据送入管道;而接受进程可从管道的另一端接收大量的数据。由于通信是采用管道实现的,所以叫管道通信。

进程通信有直接和间接两种方式。

1) 直接通信

直接通信是将消息直接发送给指定进程。因此,Send 和 Receive 原语中应指出进程名字。其调用格式如下:

Send(Who,Message)	发送消息给指定进程或一组进程
Receive(Who,Message)	从约定进程接收消息

2) 间接通信

这种通信是以信箱为媒介实现通信的,只要接收信件的进程设立一个信箱,这样,若干个进程都可以向同一个进程发送信件。因此,Send 和 Receive 原语中应给出信箱名。其调用格式如下:

Send(N,M)	将信件 M 发送到信箱 N 中
Receive(N,X)	从信箱 N 中取一封信存入 X

但有的系统还提供带标记的发送,其调用格式如下:

Send(Who,Message,tag)

其中 Tag 可以指定进程是否需要等到接收进程(取到信息)再继续运行。一般接收进程总是要等待消息到达后才继续运行。

3.2.4 管程

1. 管程的引入

用信号量和 P、V 操作来解决进程的同步与互斥问题,须在程序中的适当位置安排

P、V 操作,否则会造成死锁错误。为了解决分散编程带来的困难,1974 年和 1975 年,汉森(Brinsh Hansen)和霍尔(Hoare)提出了另一种同步机制——管程(monitor)。其基本思路是采用资源集中管理的方法,将系统中的资源用某种数据结构抽象的表示出来。由于临界区是访问共享资源的代码段,可建立一个管程管理进程提出的访问请求。

采用这种方式管理共享资源可以借助数据结构及在其上实施操作的若干过程来进行,对共享资源的申请和释放可以通过过程在数据结构上的操作来实现。

管程是由一些共享数据、一组能为并发进程执行的作用在共享数据上的操作的集合、初始代码以及存取权组成的。管程提供了一种允许多进程安全有效地共享抽象数据类型的机制。管程实现同步机制的基础是“条件结构”(condition construct)。为实现进程的互斥与同步,必须定义一些条件变量。这些条件变量只能被 wait 和 signal 操作访问。notfull. wait 操作意味着调用该操作的进程将被挂起,使另一个进程执行。而 notfull. signal 操作仅仅是启动一个被挂起的进程,如无挂起进程则 notfull. signal 操作相当于空操作,不改变 notfull 状态,这不同于 V 操作。

2. 管程的结构

每个管程都要有一个名字标识,用语言描述管程的形式如下所示。

```

Type <管程名> = monitor
  <管程变量说明>;
  define <(能被其他模块引用的) 过程名列表>;
  procedure <过程名> (<形式参数表>)
  begin
    <过程体>;
  end;
  ...
  procedure <过程名> (<形式参数表>)
  begin
    <过程体>;
  end;
begin
  <管程的局部数据初始化语句>;
end.
    
```

3. 利用管程解决生产者-消费者问题

【例 3.4】 建立一个管程 Producer-Consumer,其中包括两个过程 put(item)和 get(item)。它们分别执行将生产的消息放入缓冲池和从缓冲池中取出消息的操作,变量

count 表示缓冲池已存消息的数目。管程描述如下所示。

```
Type Producer-Consumer = monitor
  var buffer: array [ 0, ..., n-1 ] of item; {定义缓冲区}
  in, out, count: integer; {in out 为存取指针, count 为缓冲区产品个数}
  notfull, notempty: condition; {条件变量}
  procedure entry put (item)
  begin
    if count >= n then notfull. wait; {缓冲区已满}
    buffer ( in ) := nextp;
    in := (in+1) mod n;
    count = count+1;
    if notempty. queue then notempty. signal; {唤醒等待者}
  end procedure entry get ( item)
  begin
    if count <= 0 then notempty. wait; {缓冲区已空}
    nextc := buffer ( out );
    out := (out+1) mod n;
    count := count-1; {减少一个产品}
    if notfull. queue then notfull. signal; {唤醒等待者}
  end
  begin in := out := 0; count := 0; end {初始化}
```

利用管程解决生产者-消费者问题,其中生产者和消费者程序如下所示。

```
cobegin
  producer: begin
    repeat
      produce an item in nextp;
      Producer-Consumer. put ( item);
    until false;
  end
  consumer: begin
    repeat
      Producer-Consumer. get (item);
      consume the item in nextc;
    until false;
  end;
coend
```

3.2.5 进程调度

在某些操作系统中,一个作业从提交到完成需要经历高、中、低三级调度。

- **高级调度**: 又称“长调度”、“作业调度”或“接纳调度”。它决定处于输入池中的哪个后备作业可以调入主系统做好运行的准备,成为一个或一组就绪进程。系统中一个作业只须经过一次高级调度。
- **中级调度**: 又称“中程调度”或“对换调度”。它决定处于交换区中的就绪进程哪个可以调入主存,以便直接参与对 CPU 的竞争。在主存资源紧张时,为了把进程调入主存,必须将主存中处于阻塞状态的进程调至交换区,以便为调入进程腾出空间。
- **低级调度**: 又称“短程调度”或“进程调度”。它决定处于主存中的就绪进程哪个可以占用 CPU。它是操作系统中最活跃、最重要的调度程序,对系统的影响很大。
- **调度方式**: 是指当有更高优先级的进程到来时如何分配 CPU。调度方式分为可剥夺和不可剥夺两种。可剥夺式是指当有更高优先级的进程到来时,强行将正在运行的进程占用的 CPU 分配给高优先级的进程。不可剥夺式是指当有更高优先级的进程到来时,必须等待正在运行的进程自动释放占用的 CPU,然后将 CPU 分配给高优先级的进程。

常用的进程调度算法有: 先来先服务、时间片轮转、优先级调度和多级反馈调度算法。

先来先服务(FCFS)是按照作业提交或进程变为就绪状态的先后次序分配 CPU。即每当进入进程调度时,总是将就绪队列队首的进程投入运行。FCFS 算法主要用于宏观调度,其特点是: 比较有利于长作业,而不利于短作业;有利于 CPU 繁忙的作业,而不利于 I/O 繁忙的作业。

时间片轮转(round robin)的基本思路是通过时间片轮转,提高进程并发性和响应时间,从而提高资源利用率。时间片轮转算法主要用于微观调度,其设计目标是提高资源利用率。

优先级调度分为静态优先级和动态优先级两种。静态优先级: 进程的优先级是在创建时就已确定好了,直到进程终止都不会改变。动态优先级: 在创建进程时赋予一个优先级,在进程运行过程中还可以改变,以便获得更好的调度性能。

多级反馈调度算法是在时间片轮转算法和优先级算法的基础上改进的。其优点是: 照顾短进程以提高系统吞吐量,缩短平均周转时间;照顾 I/O 型进程以获得较好的 I/O 设备利用率和缩短响应时间;不必估计进程的执行时间和动态调节优先级。

下面是其算法实现。

① 设置多个就绪队列：队列 1, 队列 2, ..., 队列 n , 分别赋予不同的优先级, 队列 1 的优先级 \geq 队列 2 的优先级 \geq ... \geq 队列 n 的优先级。每个队列执行时间片的长度也不同, 规定优先级越低则时间片越长, 如逐级加倍。

② 新进程进入内存后, 先投入队列 1 的末尾, 按 FCFS 算法调度。若某进程在队列 1 的一个时间片内未能执行完, 则降低投入队列 2 的末尾, 同样按 FCFS 算法调度。如此下去, 当进程降低到最后的队列, 则按“时间片轮转”算法调度直到完成。

③ 仅当较高优先级的队列为空, 才调度较低优先级队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列, 则抢先执行新进程, 并把被抢先的进程投入原队列的末尾。

优先级的确定需要考虑如下情况。

(1) I/O 型进程：让其进入最高优先级队列, 以便能及时响应需要 I/O 交互的进程。通常执行一个小的时间片, 在该时间片内, 要求能处理完一次 I/O 请求的数据, 然后转入阻塞队列。

(2) 计算型进程：每次都执行完时间片, 进入更低级队列。最终采用最大时间片来执行, 减少调度次数。

(3) I/O 次数不多而主要是 CPU 处理的进程：在 I/O 完成后, 放回优先 I/O 请求时离开的队列, 以免每次都回到最高优先级队列后再逐次下降。

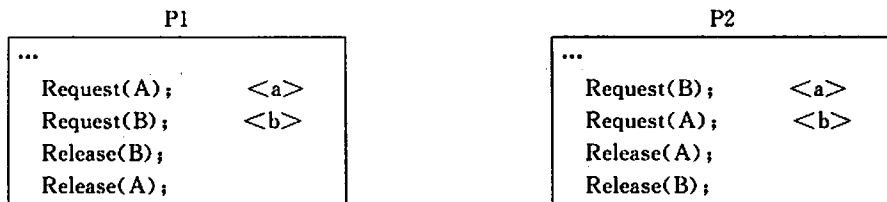
(4) 为适应一个进程在不同时间段的运行特点, I/O 完成时, 提高优先级。时间片用完时, 降低优先级。

3.2.6 死锁

计算机系统中有许多互斥资源(如磁带机、打印机和绘图仪等)或软件资源(如进程表和临界区等), 若两个进程同时使用打印机, 或同时进入临界区必然会出现问题。所谓死锁是指两个以上的进程互相都因要求对方已经占有的资源, 导致无法继续运行下去的现象。

1. 死锁举例

【例 3.5】 进程推进顺序不当引起的死锁。设系统中有一台读卡机 A, 一台打印机 B, 它们被进程 P1 和 P2 共享, 两个进程并发执行并按下列顺序请求和释放资源：



假如按 $P1 \langle a \rangle$ $P2 \langle a \rangle$ $P1 \langle b \rangle$ $P2 \langle b \rangle$ 的次序执行,则系统发生死锁。因为进程 $P1 \langle a \rangle$ 时,由于读卡机未被占用,所以请求可以得到满足;进程 $P2 \langle a \rangle$ 时,由于打印机未被占用,所以请求也可以得到满足;进程 $P1 \langle b \rangle$ 时,由于打印机被占用,所以请求得不到满足, $P1$ 等待;进程 $P2 \langle b \rangle$ 时,由于读卡机被占用,所以请求得不到满足, $P2$ 也等待。导致互相在请求对方已占有的资源,系统发生死锁。

【例 3.6】 同类资源分配不当引起死锁。若系统中有 m 个资源被 n 个进程共享,当每个进程都要求 k 个资源,而 $m < nk$ 时,即资源数小于进程所要求的总数时,可能会引起死锁。例如, $m=5, n=3, k=3$,若系统采用的分配策略是轮流地为每个进程分配资源,则第一轮系统先为每个进程分配 1 台,还剩下 2 台。第二轮系统再为 2 个进程各分配 1 台,此时,系统中已无可供分配的资源,使得各个进程都处于等待状态,导致系统发生死锁。

【例 3.7】 PV 操作使用不当引起的死锁。如图 3-9 所示,当信号量 $S1=S2=0$ 时,将发生死锁。

在图 3-9 中, $P2$ 进程从缓冲区取产品前,先执行 $P(S2)$,由于 $S2=-1$ 故 $P2$ 等待。 $P1$ 进程将产品送到缓冲区后,执行 $P(S1)$,由于 $S1=-1$ 故 $P1$ 等待。这样, $P1$ 、 $P2$ 进程都无法继续运行下去,导致系统死锁。

2. 死锁产生的原因及条件

可以看出,产生死锁的原因是资源竞争及进程推进顺序非法。当系统中的共享资源不足以同时满足多个进程的需求时,引起资源的竞争,因而导致死锁。进程推进顺序非法意指进程在运行的过程中,请求和释放资源的顺序不当,导致进程死锁。

1) 产生死锁的 4 个必要条件

- (1) 互斥条件: 进程对其要求的资源进行排他性控制,即一次只允许一个进程使用。
- (2) 请求保持条件: 零星地请求资源,即已获得部分资源后又请求资源被阻塞。
- (3) 不可剥夺条件: 进程已获得的资源在未使用完之前不能被剥夺,只能在使用完时由自己释放。
- (4) 环路条件: 发生死锁时,在进程资源有向图中必构成环路,其中每个进程占有下一个进程申请的一个或多个资源,如图 3-10 所示。



图 3-9 PV 操作引起的死锁

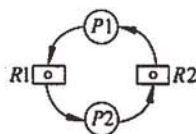


图 3-10 进程资源有向图

2) 进程资源有向图

由方框、圆圈和有向边 3 部分组成。其中方框表示资源,圆圈表示进程。请求资源:
 $\bigcirc \rightarrow \square$ 箭头由进程指向资源;分配资源: $\square \rightarrow \bigcirc$ 箭头由资源指向进程。

3. 死锁的处理

死锁的处理策略主要有 4 种。

- 鸵鸟策略: 一种不理睬策略。
- 预防策略: 破坏死锁的四个必要条件之一。
- 避免策略: 精心地分配资源,动态地回避死锁。
- 检测与解除死锁: 一旦发生死锁,系统不但能检测出,还能解除。

1) 死锁预防

死锁预防是采用某种策略,限制并发进程对资源的请求,使系统在任何时刻都不满足死锁的必要条件。预防死锁的两种策略如下。

- 预先静态分配法: 破坏“不可剥夺条件”,预先分配所需资源,保证不等待资源。该方法的问题是降低了对资源的利用率,降低进程的并发程度,有时可能无法预先知道所需资源。
- 资源有序分配法: 破坏“环路条件”,把资源分类按顺序排列,保证不形成环路。该方法存在的问题是限制进程对资源的请求,资源的排序增加系统开销。

2) 死锁避免

死锁预防是设法破坏产生死锁的四个必要条件之一,严格防止死锁的产生。死锁避免则不那么严格地限制产生死锁的必要条件。最著名的是 Dijkstra 提出的银行家算法。死锁避免算法需要很大的系统开销。

(1) 安全状态。

在避免死锁的方法中,允许进程动态地申请资源,系统在进行资源分配之前,先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态,便将资源分配给进程。否则,进程等待。

所谓安全状态,是指系统能按某种顺序如 $\langle P_1, P_2, \dots, P_n \rangle$, 为每个进程分配其所需资源,直至满足最大需求,使每个进程都可顺序完成。通常称 $\langle P_1, P_2, \dots, P_n \rangle$ 序列为安全序列。若系统不存在这样一个安全序列,则称系统处于不安全状态。

虽然并非所有不安全状态都是死锁状态,但当系统进入不安全状态后,便有可能进入死锁状态。反之,只要系统处于安全状态,系统便可避免进入死锁状态。因此,避免死锁的实质在于如何使系统不进入不安全状态。

(2) 银行家算法。

该算法对于进程发出的每一个系统可以满足的资源请求命令加以检测,如果发现分

配资源后,系统可能进入不安全状态,则不予分配。若分配资源后系统仍处于安全状态,则分配资源。与死锁预防策略相比提高了资源的利用率,但增加了系统开销。

3) 死锁检测

解决死锁的另一条途径是死锁检测方法,这种方法对资源的分配不加限制,即允许死锁产生。但系统定时地运行一个死锁检测程序,判断系统是否发生死锁,若检测到有死锁,则设法加以解除。

4) 死锁解除

死锁解除通常采用:

- 资源剥夺法,从一些进程那里强行剥夺足够数量的资源分配给死锁进程;
- 撤销进程法,根据某种策略逐个地撤销死锁进程,直到解除死锁为止。

3.2.7 线程

自 20 世纪 60 年代推出进程的概念后,操作系统中都以进程作为独立运行的基本单位。直到 20 世纪 80 年代中期,人们又推出了比进程更小的能独立运行的基本单位——线程。目的是提高系统内程序并发执行的程度,从而可进一步提高系统的吞吐量。目前不仅在新推出的操作系统中引入了线程的概念,而且在新推出的数据库管理系统和一些应用软件中,也纷纷引入线程来改善系统的性能。

传统的进程有两个基本属性:可拥有资源的独立单位,可独立调度和分配的基本单位。由于在进程的创建、撤销和切换中,系统必须为之付出较大的时空开销。因此,在系统中设置的进程数目不宜过多,进程切换的频率不宜太高,这就限制了并发程度的提高。引入线程后,将传统进程的两个基本属性分开,线程作为调度和分配的基本单位,进程作为独立分配资源的单位。用户可以通过创建线程来完成任务,以减少程序并发执行时付出的时空开销。

例如,在文件服务进程中,可设置多个服务线程,当一个线程受阻时,第二个线程可以继续运行。当第二个线程受阻时,第三个线程可以继续运行……从而显著地提高了文件系统的服务质量及系统的吞吐量。

这样,对拥有资源的基本单位,不用频繁地对其切换,从而进一步提高系统的并发程度。需要说明的是:线程是进程中的一个实体,是系统独立分配和调度的基本单位。线程基本上不拥有资源,只拥有一点运行中必不可少的资源(如程序计数器、一组寄存器和栈)。它可与同属一个进程的其他线程共享进程所拥有的全部资源。

线程可创建另一个线程,同一个进程中的多个线程可并发执行。线程也具有就绪、运行和阻塞 3 种基本状态。线程具有许多传统进程所具有的特性,故称为轻型进程(Light-Weight Process)。传统进程称之为重型进程(Heavy-Weight Process),相当于只有一个

线程的任务。

在引入了线程的操作系统中,通常一个进程都有若干个线程。可从如下 4 个方面来比较线程与进程。

- 调度: 将线程作为调度和分配的基本单位,进程作为拥有资源的基本单位。
- 并发性: 不仅进程之间可并发执行,而且同一个进程中的多个线程之间也可并发执行。
- 拥有资源: 进程是拥有资源的一个独立单位,线程不拥有系统资源,但可访问隶属于进程的资源。
- 系统开销: 在创建或撤销进程时,由于系统都要为之分配和回收资源,导致系统的开销明显地大于创建或撤销线程时的开销。

线程分为用户级线程 (User-Level Threads) 和内核支持线程 (Kernel-Supported Threads) 两类。用户级线程不依赖于内核,该类线程的创建、撤销和切换都不利用系统调用实现。某些系统中实现的是内核支持线程,这种线程依赖于内核,即无论是用户进程中的线程,还是系统中的线程,它们的创建、撤销和切换都利用系统调用实现,因此该类线程与内核有关。还有一些系统同时实现了两种类型的线程。但与线程不同的是,无论系统进程还是用户进程,在进行切换时,都要依赖于内核中的进程调度。因此,进程都是与内核有关的,是在内核支持下进行切换的。

3.3 存储管理

存储器管理的对象是主存储器(简称主存或内存)。存储器是计算机系统中的关键性资源,是存放各种信息的主要场所。特别是近几年来,系统软件和应用软件在功能及其所需存储空间等方面都在急剧膨胀。如何对存储器实施有效的管理,不仅直接影响到存储器的利用率,而且还对系统性能有重大的影响。虽然主存容量在不断扩大,但主存仍是宝贵资源,如何提高主存的利用率,扩充主存,对主存信息实现有效保护是存储器管理的主要任务。

3.3.1 基本概念

1. 存储器的结构

存储器的功能是保存数据。存储器的发展方向是高速、大容量和小体积。例如,主存在访问速度方面发展起来的有 DRAM、SDRAM 和 SRAM 等。硬盘技术在大容量方面的发展主要体现在接口标准和存储密度等方面。

存储组织的功能是在存储技术和 CPU 寻址技术许可的范围内组织合理的存储结

构,使得各层次的存储器都处于均衡的繁忙状态(如提高缓存命中率使主存读写保持繁忙),其依据是访问速度匹配、容量要求和价格等。一般存储器的结构有“寄存器-主存-外存”结构和“寄存器-缓存-主存-外存”结构(如图 3-11 所示)。

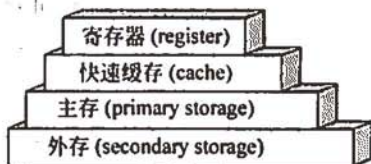


图 3-11 存储器的层次结构

- **虚拟地址**: 对程序员来说,数据的存放地址是由符号决定的,故称符号名地址,或者称为名地址,而把源程序的地址空间叫做符号名地址空间或者名空间。它从 0 号单元开始编址,并顺序分配所有符号名所对应的地址单元,所以它不是主存中的真实地址,称为相对地址、程序地址、逻辑地址或虚拟地址。
 - **地址空间**: 人们把程序中由符号名组成的空间称为地址空间。源程序经过汇编或编译后,再经过链接程序加工成程序的装配模块,即转换为相对地址编址的模块。它是以 0 为基址顺序进行编址的。相对地址也称为逻辑地址或虚地址,程序中由相对地址组成的空间叫做逻辑地间。相对地址空间通过地址再定位机构转换到绝对地址空间,绝对地址空间也叫物理地址空间。
 - **存储空间**: 简单说来,逻辑地址空间(简称地址空间)是逻辑地址的集合,物理地址空间(简称存储空间)是物理地址的集合。
2. 地址重定位
- **重定位**: 程序的逻辑地址被转换成主存的物理地址的过程称为地址重定位。可执行文件装入主存时需要解决可执行文件中地址(指令和数据)和主存地址的对应关系。由操作系统中的装入程序 Loader 和地址重定位机构来完成。地址重定位分为静态地址重定位和动态地址重定位。
 - **静态重定位**: 是指在程序装入主存时已经完成了逻辑地址到物理地址的变换,在程序的执行期间不会再次发生变化。静态地址重定位的优点是无须硬件地址变换机构的支持,它只要求程序本身是可重定位的,它只对那些要修改的地址部分具有某种标识,由专门设计的程序来完成。早期的操作系统中多数都采用这种方法。静态重定位的缺点是必须给作业分配一个连续的存储区域,在作业的执行期间不能扩充存储空间,也不能在主存中移动,多个作业也难以共享主存中的同一程序副本和数据。
 - **动态重定位**: 在程序运行期间完成逻辑地址到物理地址的变换。其实现依赖于硬件地址变化机构,如基址寄存器 BR。动态地址重定位的优点: 程序在执行期间可以换入和换出主存,可以解决主存紧张问题;可以在主存中移动,把主存中的碎片集中起来,以便充分利用空间;不必给程序分配连续的主存空间,可以较好

的利用较小的主存块;可以实现共享。

3. 存储管理的功能

- 主存储器的分配和回收: 主存分配的主要任务是为每一道程序分配主存空间。
- 提高主存储器的利用率: 减少碎片(也称零头), 允许多道程序动态共享主存。
- 存储保护: 主存保护的任务是确保每道程序都在自己的主存空间运行, 互不干扰。
- 主存扩充: 主存扩充的任务是从逻辑上扩充主存容量, 使用户认为系统拥有的主存空间远比其实际的主存空间(RAM)大得多。

3.3.2 分区存储管理

按分区的划分方式不同可以分为固定分区、可变分区和可重定位分区。

1. 固定分区

固定分区是一种静态分区方式。在系统生成时已将主存划分为若干个分区, 每个分区的大小可不等。假如系统将主存的用户可用空间分了 5 个区, 分别为 8KB、32KB、32KB、50KB 和 150KB, 如图 3-12 所示。操作系统通过主存分配情况表管理主存。这种方法的突出问题是已分配区中存在未用空间, 原因是程序或作业的大小不可能刚好等于分区的大小, 故造成了空间的浪费。通常将已分配分区内的未用空间叫做零头或内碎片。

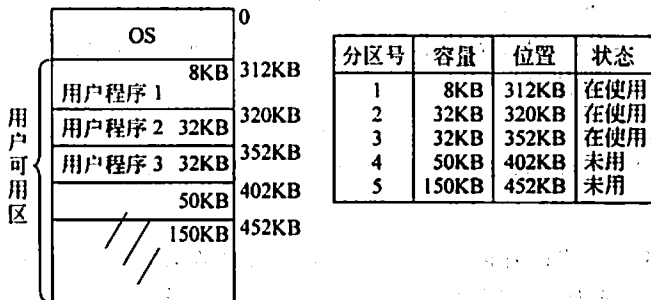


图 3-12 固定分区

2. 可变分区

可变分区是一种动态分区方式, 存储空间的划分是在作业装入时进行的, 故分区的个数是可变的, 分区的大小刚好等于作业的大小。可变分区分配需要两种管理表格: 已分配表, 记录已分配分区的情况; 未分配表, 记录未分配分区的情况。请求和释放分区主要有如下 4 种算法。

1) 最佳适应算法

假设系统中有 $X_1, X_2, \dots, X_i, \dots, X_n$ 个空白区(自由区), 每当用户申请一个空间时, 将从这 n 个空白区中找到一个最接近用户需求的分区。这种算法能保留较大的空白区。但缺点是空闲区不可能刚好等于用户要求的区, 所以必然要将分区一分为二。随着系统不断地分配和释放空间, 产生的小分区小到可能会无法再继续分配。人们将这样的无用小分区称之为外碎片。

2) 最差适应算法

系统总是将用户作业装入最大的空白分区。这种算法将一个最大的分区一分为二, 所以剩下的空白区通常也大, 不容易产生外碎片。

3) 首次适应算法

每当用户作业申请一个空间时, 系统总是从主存的低地址开始选择一个能装入作业的空白区。当用户释放空间时, 该算法更易实现相邻的空白区合并。

4) 循环首次适应算法

与首次适应算法的不同之处是, 每次分配都是从刚分配的空白区开始寻找一个能满足用户要求的空白区。

引入可变分区后虽然主存分配更灵活, 也提高了主存利用率, 但由于系统在不断地分配和回收中, 必定会出现一些不连续的小的空闲区, 尽管这些小的空闲区的总和超过某一个作业要求的空间, 但由于不连续而无法分配, 产生了碎片。解决碎片的方法是拼接(或称紧凑), 即向一个方向(例如向低地址端)移动已分配的作业, 使那些零散的小空闲区在另一方向连成一片。分区的拼接技术一方面要求能够对作业进行重定位; 另一方面系统在拼接时要耗费较多的时间。

假设主存有 3 个未分配区 32KB、128KB 和 136KB, 某作业需要 150KB 的空间。此时, 尽管系统总的空闲空间能满足要求, 但还是不能为作业分配。

3. 可重定位分区

可重定位分区是解决碎片问题的简单而又行之有效的方法。基本思想: 移动所有已分配好的分区, 使之成为连续区域。如同队列有一个队员出列, 指挥员叫大家“靠拢”一样。分区“靠拢”的时机: 当用户请求空间得不到满足时或某个作业执行完毕时。由于靠拢是要代价的, 所以通常是在用户请求空间得不到满足时进行。

例如, 系统主存使用情况如图 3-13(a) 所示。当用户(作业 7)申请 256KB 主存时, 由于没有满足要求的空白区, 所以系统需要执行靠拢, 如图 3-13(b) 所示。为用户分配后的情况如图 3-13(c) 所示。但当移动已分配的分区会导致地址发生变化, 所以有地址重定位的问题。解决程序重定位问题可以有两种方法。

(1) 使用模块装入程序, 将该程序的装配模块重新装入指定位置, 使程序重新开始执

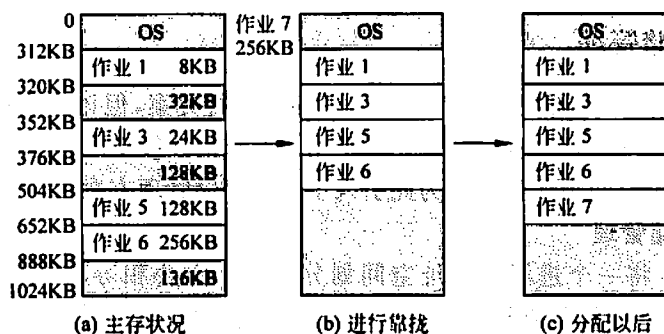


图 3-13 可重定位分区分配

行。这样做不仅浪费 CPU 的时间,而且有的程序一旦执行不能重新开始,否则可能会引起系统混乱。

(2) 采用动态重定位技术,即利用一个重定位寄存器(再定位寄存器),自动修改访问存储器的地址。

4. 分区的保护问题

分区保护通常采用上界/下界寄存器、基址/限长寄存器的方法。采用上界/下界寄存器保护法时,上界寄存器中存放的是作业的装入地址,下界寄存器装入的是作业的结束地址,形成的物理地址必须满足:

上界寄存器 \leq 物理地址 \leq 下界寄存器

采用基址/限长寄存器保护法时,基址寄存器中存放的是作业的装入地址,限长寄存器装入的是作业的长度,形成的物理地址必须满足:

基址寄存器 \leq 物理地址 $<$ 基址寄存器 + 限长寄存器

可变分区地址映射如图 3-14 所示。

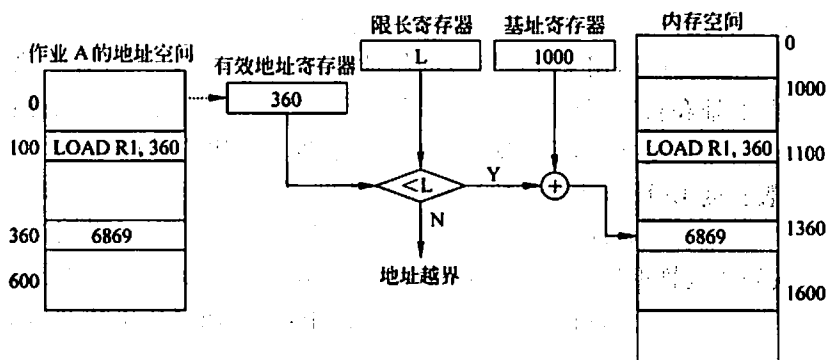


图 3-14 可变分区地址映射

3.3.3 分页存储管理

尽管分区管理方案是解决多道程序共享主存的可行方案,但该方案的主要问题是用户程序必须装入连续的地址空间中,若无满足用户要求的连续空间,需要进行分区靠拢操作,这是以耗费系统时间为代价的。为此引入了分页存储管理方案。

1. 纯分页存储管理

- **分页原理:** 将一个进程的地址空间划分成若干大小相等的区域,称为页。相应地,将主存空间划分成与页相同大小的若干物理块,称为块或页框。在为进程分配主存时,将进程中若干页分别装入多个不邻接的块中。
- **地址结构:** 分页系统的地址结构如图 3-15 所示,它由两部分组成:前一部分为页号 P ;后一部分为偏移量 W ,即页内地址。图中的地址长度为 32 位,其中 0~11 位为页内地址(每页的大小为 4KB),12~31 位为页号,所以允许地址空间的大小最多为 1MB 个页。

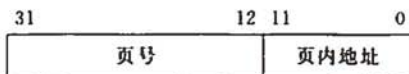


图 3-15 分页地址结构

- **页表:** 在将进程的每一页离散地分配到主存的多个物理块中后,系统应保证能在主存中找到每个页面所对应的物理块。为此,系统为每个进程建立一张页面映射表,简称页表(如图 3-16 所示)。每个页在页表中占一个表项,记录该页在主存中对应的物理块号。进程执行时,通过查找页表,就可以找到每页所对应的物理块号。可见,页表的作用是实现从页号到物理块号的地址映射。

地址变换机构的基本任务是利用页表把用户程序中的逻辑地址变换成主存中的物理地址,实际上就是将用户程序中的页号变换成主存中的物理块号。为了实现地址变换功能,系统中设置页表寄存器,用来存放页表的始址和页表的长度。在进程未执行时,每个进程对应页表的始址和长度存放在进程的 PCB 中。当该进程被调度时,就将它们装入页表寄存器。在进行地址变换时,系统对页号与页表长度进行比较,如果页号大于等于页表寄存器中的页表长度 L (页号从 0 开始),则访问越界,产生越界中断。若未出现越界,则根据页表寄存器中的页表始址和页号计算出该页在页表项中的位置,得到该页的物理块号,将此物理块号装入物理地址寄存器中。与此同时,将有效地址(逻辑地址)寄存器中的页内地址直接装入物理地址寄存器的块内地址字段中,这样便完成了从逻辑地址到物理地址的变换。

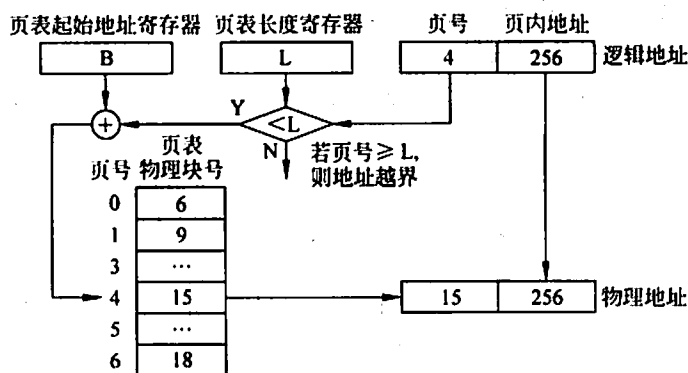


图 3-16 页式存储管理的地址映射

2. 快表

从地址映射的过程可以发现,页式存储管理至少需要访问主存两次。第一次是访问页表,得到数据的物理地址;第二次是存取数据。若数据是间接地址,还需要再进行地址变换。为了提高访问主存的速度,可以在地址映射机构中增加一组高速寄存器,用来保存页表。这种方法需要大量的硬件开销,经济上是不可行的。另一种方法是在地址映射机构中增加一个小容量的联想存储器。联想存储器由一组高速存储器组成,称之为快表,用来保存当前访问频率高的少数活动页的页号及相关信息。在快表中,除了逻辑页号、物理块号外,还要增加特征位,表示该行是否为空。增加访问位,表示最近该页是否被访问过,目的是为了淘汰那些长时间未访问过或用的很少或不用的页面。

联想存储器存放的只是当前进程最活跃的少数几页,因此,当用户程序要访问数据时,根据该数据所在逻辑页号,在联想存储器中找出对应的物理页号,然后与页内地址拼接形成物理地址。若找不到相应的逻辑页号,则地址映射仍通过主存的页表进行。得到物理地址后,须将物理块号填入联想存储器的空闲单元中。若无空闲单元,则根据淘汰算法淘汰某一行,再填入新得到的页号。事实上,查找联想存储器和查找主存页表是并行进行的。一旦在联想存储器中找到相符的逻辑页号,就停止查找主存页表。

3. 两级页表机制

80386 的逻辑地址有 2^{32} 个,若页面大小为 4KB(2^{12} B),则页表项达 1M 个。每个页表项占用 4B,故每个进程的页表占用 4MB 主存空间,并且还要求是连续的,显然这是不现实的。在 80386 中,为了减少页表所占用的连续主存空间,采用了两级页表机制。基本方法是将页表进行分页,每个页面的大小与主存物理块的大小相同,并为它们编号。可以离散地将各个页面分别存放在不同的物理块中。为此需要建立一张页表,称为外层页表(页

表目录),即第一级页表,其中的每个表目存放某个页表的物理地址。第二级是页表,其中的每个表目存放的是页的物理块号。

两级页表的逻辑地址结构和两级页表的地址变换机构如图 3-17 所示。

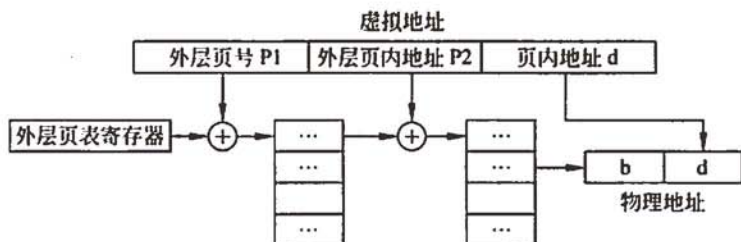


图 3-17 两级页表的地址变换机构

3.3.4 分段存储管理

1. 引入分段存储管理方法的原因

- 便于编程: 通常用户常常把自己的作业按照逻辑关系划分成若干个段,每个段都有自己的名字,且都从零开始编址。这样,用户程序在执行中可用段名和段内地址进行访问。例如,LOAD 1,[A] | <D> 这条指令的含义是将分段 A 中 D 单元内的值读入寄存器 1。
- 分段共享: 在实现程序和数据的共享时,常常以信息的逻辑单位为基础,但分页系统中的每一页只是存放信息的物理单位,其本身没有完整的意义,因而不便于实现信息的共享,而段却是信息的逻辑单位,有利于信息的共享。
- 分段保护: 信息保护是对具有完整意义的逻辑单位(段)进行保护。
- 动态连接: 通常,一个源程序经过编译后可形成若干个目标程序,还需要经过链接,形成可执行代码后才能运行,这种在装入时进行的链接称为静态链接。动态链接是指在作业运行之前,并没有把目标程序段都链接起来,而是先将主程序对应的目标程序装入主存并启动运行,当运行过程中需要调用某段代码时,再将该段(目标程序)调入主存并链接起来。所以,动态链接是以段为基础的。
- 动态增长: 在实际系统中,有些数据段往往会不断地增长,而事先却无法知道数据段会增长到多大,分段存储管理方式可以较好地解决这个问题。

2. 分段的基本原理

在分段存储管理方式中,作业的地址空间被划分为若干个段。每个段是一组完整的逻辑信息,如有主程序段、子程序段、数据段及堆栈段等。每个段都有自己的名字,都是从

零开始编址的一段连续的地址空间,各段长度不等。分段系统的地址结构如图 3-18 所示,逻辑地址由段号(名)和段内地址两部分组成。在该地址结构中,允许一个作业最多有 64K 段,每个段的最大长度为 64KB。

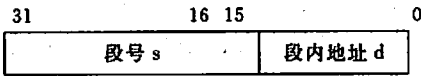


图 3-18 分段的地址结构

在分段式存储管理系统中,为每个段分配一个连续的分区,而进程中的各个段可以离散地分配到主存的不同分区中。系统为每个进程建立一张段映射表,简称为“段表”。每个段在表中占有一项,其中记录了该段在主存中的起始地址(又称为“基址”)和段的长度,如图 3-19 所示。进程在执行时,通过查段表找到每个段对应的主存区。可见,段表实现了从逻辑段到物理主存区的映射。

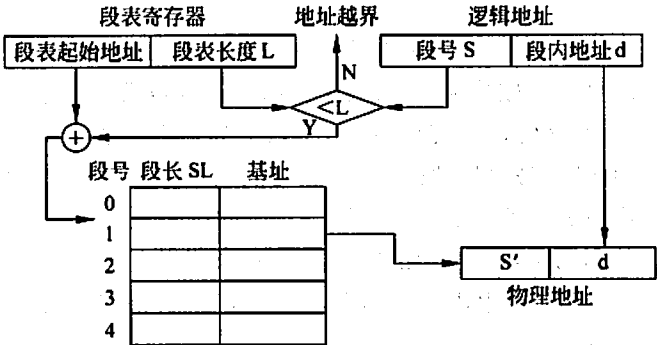


图 3-19 段式存储管理的地址变换机构

为了实现从逻辑地址到物理地址的变换功能,系统中设置了段表寄存器,用于存放段表始址和段表长度。在进行地址变换时,系统对逻辑地址中的段号 S 与段表长度 L 进行比较。若 $S \geq L$,表示段号太大,访问越界,于是产生越界中断信号。若未越界,则根据段表的始址和该段的段号,计算出该段对应段表项的位置,从中读出该段在主存中的起始地址,然后再检查段内地址 d 是否超过该段的段长 SL。若超过,即 $d \geq SL$,同样发出越界中断信号。若未越界,则将该段的基址 S' 与段内地址 d 相加,得到要访问的主存物理地址。

由于段是信息的逻辑单位,因此分段系统的一个突出优点是易于实现段的共享,即允许若干个进程共享一个或多个段,而且对段的保护也十分简单。在分页系统中,虽然也能实现程序和数据的共享,但远不如分段系统来得方便。

3.3.5 段页式存储管理

分页和分段存储管理方式都各有其优缺点。如果对两种存储管理方式“各取所长”，则可以形成一种新的存储管理方式的系统——段页式系统。这种新系统既具有分页系统能有效地提高主存利用率的优点，又具有分段系统能很好地满足用户需要的长处，显然是一种比较有效的存储管理方式。

段页式系统的基本原理是先将整个主存划分成大小相等的存储块（页架），将用户程序按程序的逻辑关系分为若干个段，并为每个段赋予一个段名。再将每个段划分成若干页，以页架为单位离散分配。在段页式系统中，其地址结构由段号、段内页号和页内地址 3 部分组成。作业地址空间的结构如图 3-20 所示。

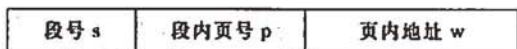


图 3-20 段页式管理的地址结构

在段页式系统中，为了实现从逻辑地址到物理地址的变换，系统中必须同时配置段表和页表。由于段中的页是离散分配的，段表中的内容不再是段的主存始址和段长，而是页表始址和页表长度。在段页式系统中，有一个段表寄存器，存放段表始址和段表长度 TL。在进行地址变换时，首先利用段号 S ，将它与段表长度 TL 进行比较。若 $S < TL$ ，表示未越界。于是利用段表始址和段号求出该段对应的段表项在段表中的位置，从中得到该段的页表始址。并利用逻辑地址中的段内页号 P 来获得对应页的页表项位置，从中读出该页所在的物理块号 b 。再用块号 b 和页内地址构成物理地址。段页式系统中的地址变换机构如图 3-21 所示。

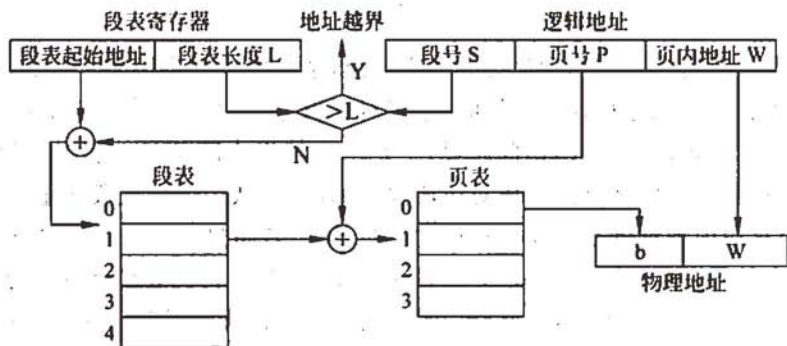


图 3-21 段页式存储管理的地址变换机构

3.3.6 虚拟存储管理

1. 虚拟存储器的引入

1) 局部性原理

早在 1968 年 P. Denning 就指出过,程序在执行时将呈现出局部性规律,即在一段时间内,程序的执行仅局限于某个部分。相应地,它所访问的存储空间也局限于某个区域内。程序的局限性表现在时间局限性和空间局限性两个方面。

- 时间局限性:如果程序中的某条指令一旦执行,则不久的将来该指令可能再次被执行。如果某个存储单元被访问,则不久以后该存储单元可能再次被访问。产生时间局限性的典型原因是程序中存在大量的循环操作。
- 空间局限性:一旦程序访问了某个存储单元,则在不久的将来,其附近的存储单元也最有可能被访问。即程序在一段时间内访问的地址可能集中在一定的范围内,其典型原因是程序是顺序执行的。

2) 虚拟存储器的定义

根据局部性原理,一个作业在运行之前,没有必要把作业全部装入主存。仅将那些当前需要运行的部分页面或段先装入主存便可启动运行,其余部分可暂时留在磁盘上。

程序运行时,如果需要访问的页(段)已调入主存,便可继续执行下去。但如果程序所要访问的页(段)尚未调入主存(称为缺页或缺段),此时程序应利用操作系统提供的请求调页(段)功能,将它们调入主存,以使进程能继续执行下去。

如果此时主存已满,无法再装入新的页(段),则还须再利用页(段)的置换功能,将主存中暂时不用的页(段)调出至磁盘上。腾出足够的主存空间后,再将所要访问的页(段)调入主存,使程序继续执行下去。这样,便可使一个大的用户程序在较小的主存空间中运行。也可使主存中同时装入更多的进程并发执行。从用户角度看,系统具有的主存容量比实际主存容量大得多,人们把这样的存储器称为虚拟存储器。

虚拟存储器具有请求调入功能和置换功能,能仅把作业的一部分装入主存运行。是从逻辑上对主存容量进行扩充。其逻辑容量由主存和外存容量之和以及 CPU 可寻址的范围决定,其运行速度接近于主存速度,成本也下降。可见,虚拟存储技术是一种性能非常优越的存储器管理技术,故被广泛地应用于大、中、小和微型机中。

3) 虚拟存储器的实现

- 请求分页系统:它是在分页系统的基础上,增加了请求调页功能和页面置换功能形成的页式虚拟存储系统。它允许只装入若干页的用户程序和数据(而非全部程序),就可以启动运行,以后再通过调页功能和页面置换功能,陆续把将要使用的页面调入主存。同时把暂不运行的页面置换到外存上,置换时以页面为单位。

- **请求分段系统**：它是在分段系统的基础上，增加了请求调段和分段置换功能形成的段式虚拟存储系统。它允许只装入若干段（而非全部段）的用户程序和数据，就可以启动运行。以后再通过调段功能和置换功能将不运行的段调出，同时调入将要运行的段。置换时以段为单位。
 - **请求段页式系统**：它是在段页式系统的基础上，增加了请求调页和页面置换功能形成的段页式虚拟存储系统。
- 4) 虚拟存储器的特征
- **离散性**：指在主存分配时采用离散的分配方式。它是虚拟存储器最基本的特征。
 - **多次性**：指一个作业多次调入主存运行，即在作业运行时没有必要将其全部装入，只须将当前要运行的那部分程序和数据装入主存即可。多次性是虚拟存储器最重要的特征。
 - **对换性**：指允许在作业的运行过程中在主存和外存的对换区之间换进、换出。
 - **虚拟性**：指能够从逻辑上扩充主存容量，使用户看到的主存容量远大于实际主存容量。

2. 请求分页中的硬件支持

请求分页是目前常用的一种虚拟存储器方式。

1) 请求分页的页表机制

它是在纯分页的页表机制上形成的，由于只将应用程序的一部分调入主存，还有一部分仍在磁盘上，故需在页表中再增加若干项，如状态位、访问字段和辅存地址等供程序（数据）在换进、换出时参考。

2) 缺页中断机构

在请求分页系统中，每当所要访问的页面不在主存时，便要产生一缺页中断，请求操作系统将缺页调入主存。与一般中断的主要区别在于：缺页中断在指令执行期间产生和处理中断信号，而一般中断在一条指令执行完后检查和处理中断信号。缺页中断返回到该指令的开始重新执行该指令，而一般中断返回到该指令的下一条指令执行。一条指令在执行期间可能产生多次缺页中断。

3) 地址变换机构

请求分页系统中的地址变换机构是在分页系统的地址变换机构的基础上形成的。为实现虚拟存储器而增加了某些功能，如产生和处理缺页中断，从主存中换出一页等。

3. 页面置换算法

请求分页是在纯分页系统的基础上，增加了请求调页功能和页面置换功能形成的页式虚拟存储系统，是目前常用的一种虚拟存储器方式。在进程运行过程中，如果发生缺页，此时主存中又无空闲块时，为了保证进程能正常运行，必须从主存中调出一页程序或

数据并送到磁盘的对换区。但究竟将哪个页面调出,需要根据一定的页面置换算法确定。置换算法的好坏将直接影响系统的性能,不适当的算法可能会导致系统发生“抖动”(thrashing)。即刚换出的页很快又被访问,需重新调入,导致系统频繁地更换页面,以至于进程在运行中把大部分时间花费在页面置换上,这种现象称之为系统发生了“抖动”(也称颠簸)。请求分页系统的核心问题是选择合适的页面置换算法。常用的页面置换算法如下所述。

1) 最佳(optimal)置换算法

它是一种理想化的算法,性能最好,但在实际上难于实现。其目标是选择把那些永不使用的,或者是长时间内不再访问的页面置换出去。但要确定哪一个页面是未来最长时间内不再访问的,很难估计。所以该算法通常只是用来评价其他算法。

【例 3.8】 假定系统为某进程分配了 3 个物理块,进程访问页面的顺序为: 0,1,6,5,7,4,7,3,5,4,7,4,5,6,5,7,6,0,7,6。如图 3-22 所示,进程运行时先将 0、7、6 3 个页面装入主存。当进程访问页面 5 时,产生缺页中断。根据最佳置换算法,页面 0 将在第 18 次才被访问,是 3 页中最久不会被访问的页面,所以被淘汰。接着访问页面 7 时,发现已在主存中,因而不会产生缺页中断,依此类推。从图可以看出,采用最佳置换算法,只发生了 6 次页面置换。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0	0	0
		7	7	7	7	7	7	3	3	3	7	7	7	7	7	7	7	7	7	7
			6	6	6	4	4	4	4	4	4	4	4	6	6	6	6	6	6	6
缺页	×	×	×	×		×		×			×			×				×		

图 3-22 最佳置换算法

2) 先进先出(FIFO)置换算法

该算法总是淘汰最先进入主存的页面,即选择在主存中驻留时间最长的页面予以淘汰。该算法实现简单,只需把一个进程已调入主存的页面,按先后次序连接成一个队列,并设置一个指针即可。它是一种最直观,性能最差的算法。它有贝莱迪(Belady)异常现象:对页面访问序列 A B C D A B E A B C D E,当分配的物理块从 3 块增加到 4 块时,缺页次数反而增加。

【例 3.9】 假定系统中某进程访问页面的顺序如例 3.8 所示,利用 FIFO 算法进行页面置换的结果如图 3-23 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	6	5	7	4	4	3	5	3	7	4	5	6	6	7	7	0	0	0
		7	0	6	5	7	7	4	3	5	3	7	4	5	5	6	6	7	7	7
			7	0	6	5	5	7	4	4	5	3	7	4	4	5	5	6	6	6
缺页	×	×	×	×	×	×		×	×		×	×	×	×		×		×		

图 3-23 先进先出 FIFO 算法

从图 3-23 中可以看出,共发生了 11 次面置换,缺页次数 14 次。

3) 最近最久未使用置换算法(least recently used,LRU)

该算法选择最近最久未使用的页面予以淘汰。系统在每个页面设置一个访问字段,用以记录这个页面自上次访问以来所经历的时间 T。当要淘汰一个页面时,选择 T 最大的页面。但在实现时需要硬件的支持(寄存器或栈)。

【例 3.10】 假定系统中某进程访问页面的顺序如例 3.8 所示,利用 LRU 算法进行页面置换的结果如图 3-24 所示。

访问页面	0	7	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
物理块	0	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7	6
		7	0	6	5	7	4	7	3	5	4	7	4	5	6	5	7	6	0	7
			7	0	6	5	5	4	7	3	5	5	7	4	4	6	5	7	6	0
缺页	×	×	×	×	×	×		×	×	×	×			×		×		×		

图 3-24 最近最久未使用 LRU 算法

从图中可以看出,共发生了 10 次面置换,缺页次数 13 次。

4) 最近未用置换算法(not used recently,NUR)

将最近一段时间未引用过的页面换出。这是一种 LRU 的近似算法。该算法为每个页面设置一位访问位,将主存中的所有页面都通过链接指针连接成一个循环队列。当某页被访问时,其访问位置“1”。在选择一页淘汰时,检查其访问位,如果是“0”,则选择该页换出。若为“1”,则重新置为“0”,暂不换出该页。然后在循环队列中检查下一个页面,直到访问位为“0”的页面为止。由于该算法只有一位访问位,只能用它表示该页是否已经使用过,而置换时是将未使用过的页面换出去,所以该算法被称为最近未用算法。

4. 工作集

程序在运行中产生的缺页情况会影响程序的运行速度及系统性能,而缺页率的高低又与每个进程占用的物理块数目有关。究竟应该为每个进程分配多少个物理块,才能把

缺页率保持在一个合理的水平上。否则会因为进程频繁地从辅存请求页面,而出现“颠簸”(也称抖动)现象。所谓工作集是指在某段时间间隔(Δ)里,进程实际要访问的页面的集合。工作集的理论是在 1968 年由 Denning 提出来的,他认为,虽然程序只需有少量的几页在主存就可以运行,但为了使程序能够有效地运行,较少地产生缺页,就必须使程序的工作集驻留在主存中。把某进程在时间 t 的工作集记为 $w(t, \Delta)$,把变量 Δ 称为工作集“窗口尺寸”(windows size)。正确选择工作集窗口(Δ)的大小,对存储器的有效利用和系统吞吐量的提高,都将产生重要的影响。

程序运行时对页面的访问是不均匀的,即在某段时间内的访问往往仅局限于较少的几个页面。如果能够预知程序在某段时间间隔内要访问哪些页面,并能将它们提前调入主存,将会大大地降低缺页率,从而减少置换工作,提高 CPU 的利用率。当每个工作集都已达到最小值时,虚存管理程序跟踪进程的缺页数量,根据主存中自由页面数量可以适当增加其工作集的大小。

3.4 设备管理

设备管理是操作系统中最繁杂而且与硬件紧密相关的部分。设备管理不但要管理实际执行 I/O 操作的设备(如磁盘机、打印机),还要管理诸如设备控制器、DMA 控制器、中断控制器和 I/O 处理机(通道)等支持设备。设备管理包括各种设备分配、缓冲区管理和物理 I/O 设备操作,通过管理达到提高设备利用率和方便用户的目的。

3.4.1 设备管理概述

不同的计算机系统均使用大量不同类型的设备,这些设备的特点也完全不同。因此,设备管理是操作系统设计中最杂乱无序的领域。

设备是计算机系统与外界交互的工具,具体负责计算机与外部的输入输出工作,所以常称为外部设备,简称外设。在计算机系统中将负责管理输入输出机构称为输入输出系统。因此,输入输出系统由设备、控制器、通道(具有通道的计算机系统)、总线和输入输出软件组成。

1. 设备的分类

现代计算机系统都配有各种各样的设备,如打印机、显示器、绘图仪、扫描仪、键盘和鼠标等。但是设备可以有各种不同的分类方式。

1) 按数据组织分类

(1) 块设备(block device):指以数据块为单位来组织和传送数据信息的设备。这类设备用于存储信息,如磁盘和磁带等。它属于有结构设备。典型的块设备是磁盘,每个盘

块的大小为 512 字节~4KB,磁盘设备的基本特征是:

- 传输速率较高,通常每秒钟为几兆位;
- 它是可寻址的,即可随机地读写任意一块;
- 磁盘设备的输入输出采用 DMA 方式。

(2) 字符设备(character device):指以单个字符为单位来传送数据信息的设备。这类设备一般用于数据的输入和输出,如交互式终端和打印机等。它属于无结构设备。字符设备的基本特征是:

- 传输速率较低;
- 不可寻址,即不能指定输入时的源地址或输出时的目标地址;
- 字符设备的输入输出常采用中断驱动方式。

2) 从资源分配角度分类

(1) 独占设备:指在一段时间内只允许一个用户(进程)访问的设备,大多数低速的 I/O 设备,如用户终端和打印机等均属于这类设备。因为独占设备属于临界资源,所以多个并发进程必须互斥地进行访问。

(2) 共享设备:指在一段时间内允许多个进程同时访问的设备。显然,共享设备必须是可寻址的和可随机访问的设备。典型的共享设备是磁盘。共享设备不仅可以获得良好的设备利用率,而且是实现文件系统和数据库系统的物质基础。

(3) 虚拟设备:指通过虚拟技术将一台独占设备变换为供多个用户(进程)共享的逻辑设备。一般可以利用假脱机技术(spooling 技术)实现虚拟设备。Spooling 技术在后续内容中介绍。

3) 按数据传输率分类

(1) 低速设备:指传输速率为每秒钟几字节到数百字节的设备。典型的设备有键盘、鼠标及语音的输入等;

(2) 中速设备:指传输速率在每秒钟数千字节至数万字节的设备。典型的设备有行式打印机和激光打印机等;

(3) 高速设备:指传输速率在数十万字节至数兆字节的设备。典型的设备有磁带机、磁盘机和光盘机等。

4) 其他分类方法

按输入输出对象分:分为人机通信和机机通信设备。按是否可交互分:可分为非交互设备,如机机通信设备、外存和卡带机等;交互设备,如终端。

输入输出设备的种类繁多,从 OS 观点来看,其重要的性能指标有数据传输速率、数据的传输单位以及设备的共享属性等。

2. 设备管理的目标与任务

1) 设备管理的目标

设备管理的目标主要是如何提高设备的利用率,为用户提供方便统一的界面。提高设备的利用率,就是提高 CPU 与输入输出设备之间的并行操作程度。利用的主要技术有中断技术、DMA 技术、通道技术和缓冲技术。为用户提供方便、统一的界面。所谓方便是指用户能独立于具体设备的复杂物理特性之外而方便地使用设备。所谓统一是指对不同的设备尽量使用统一的操作方式,例如,各种字符设备用一种输入输出操作方式。这就要求用户操作的是简便的逻辑设备,而具体的 I/O 物理设备由操作系统实现,这种性能常常被称为设备的独立性。

2) 设备管理的任务

设备管理的任务是保证在多道程序环境下,当多个进程竞争使用设备时,按一定策略分配和管理各种设备,控制设备的各种操作,完成输入输出设备与主存之间的数据交换。设备管理的主要功能如下所述。

- 动态地掌握并记录设备的状态。在设置有通道的系统中,还应掌握通道和控制器的使用状态。
- 设备分配和释放:设备管理程序按照一定的策略把某个 I/O 设备及其相应的设备控制器和通道分配给某一用户(进程),以保证在输入输出设备和 CPU 之间有传输信息的通路。将未分配到设备(包括控制器和通道)的进程插入等待队列。
- 缓冲区管理:为了解决 CPU 与 I/O 设备之间速度不匹配的矛盾,在它们之间配置了缓冲区。设备管理程序要负责管理缓冲区的建立、分配和释放。
- 实现物理 I/O 设备的操作:对于具有通道的计算机系统,设备管理程序根据用户提出的 I/O 请求,生成相应的通道程序并提交给通道,然后用专门的通道指令启动通道,对指定的设备进行 I/O 操作,并能响应通道的中断请求。对于未设置通道的系统,设备管理程序直接驱动设备进行 I/O 操作。
- 提供设备使用的用户接口:命令接口和编程接口。
- 设备的访问和控制:包括并发访问和差错处理。
- I/O 缓冲和调度:目的是提高 I/O 访问效率。

3.4.2 I/O 软件

设备管理软件的设计水平决定了设备管理的效率。I/O 设备管理软件的结构的基本思想是分层构造,也就是说,把设备管理软件组织成一系列层次。其中低层与硬件相关,它把硬件与较高层次的软件隔离开来。而最高层的软件则向应用提供一个友好的、清晰而统一的接口。

设计 I/O 软件的主要目标是设备独立性和统一命名。I/O 软件独立于设备,可以提高设备管理软件的设计效率。当输入输出设备更新时,没有必要重新编写全部设备驱动程序。在实际应用中我们也可以看到,在常用的操作系统中,只要安装了相应的设备驱动程序,就可以很方便地安装好新的输入输出设备。甚至不必重新编译就能将设备管理程序移到他处执行。

I/O 设备管理软件一般分为 4 层:中断处理程序、设备驱动程序、与设备无关的系统软件 and 用户级软件。至于一些具体分层细节上的处理,是依赖于系统的,没有严格的划分,只要有利于设备独立这一目标,可以为提高效率而设计不同的层次结构。

1. 中断处理程序

一旦 CPU 响应中断,转入中断处理程序,系统就开始进行中断处理。中断处理过程如下:

① CPU 检查响应中断的条件是否满足。CPU 响应中断的条件是:有来自中断源的中断请求,CPU 允许中断。如果中断响应条件不满足,则中断处理无法进行。

② 如果 CPU 响应中断,则 CPU 关中断,使其进入不可再次响应中断的状态。

③ 保存被中断进程的现场。为了在中断处理结束后能使进程正确地返回到中断点,系统必须保存当前处理状态字 PSW 和程序计数器 PC 等信息。这些信息通常保存在特定堆栈或硬件寄存器中。

④ 分析中断原因,调用中断处理子程序。在多个中断请求同时发生时,处理优先级最高的中断源发出的中断请求。在系统中,为了处理上的方便,通常都是针对不同的中断源编制不同的中断处理子程序(陷阱处理子程序)。这些子程序的入口地址(或陷阱指令的入口地址)存放在主存的特定单元中。需要说明的是,不同的中断源也对应着不同的处理器状态字 PSW。这些不同的 PSW 被放在相应的主存单元中,与中断处理子程序入口地址一起构成中断向量。显然,根据中断或陷阱的种类,系统可从中断向量表中迅速找到该中断响应的优先级、中断处理子程序(或陷阱指令)的入口地址和对应的 PSW。

⑤ 执行中断处理子程序。有些系统通过陷阱指令向当前执行进程发出软中断信号后,调用对应的处理子程序执行。

⑥ 退出中断,恢复被中断进程的现场或调度新进程占据 CPU。

⑦ 开中断,CPU 继续执行。

2. 设备驱动程序

设备驱动程序是直接同硬件打交道的软件模块。一般而言,设备驱动程序的任务是接受来自与设备无关的上层软件的抽象请求,进行与设备相关的处理。

设备驱动程序最突出的特点是,它与 I/O 设备的硬件结构密切联系。设备驱动程序基本上是依赖于设备的代码。设备驱动程序是操作系统底层中惟一知道各种输入输出设

备的控制器细节及其用途的部分。例如,只有磁盘驱动程序具体了解磁盘的区段、磁道、柱面、磁头、磁臂的运动、交错访问系数、马达驱动器和磁头定位次数,以及所有保证磁盘正常工作的机制,其他软件根本不过问这些硬件操作的细节。

不同的操作系统中对设备驱动程序结构的要求是不同的。一般而言,在操作系统的相关文档中,都有对设备驱动程序结构方面的统一要求。对于某一类设备而言,采用通用的设备驱动程序,还是采用专用的设备驱动程序,取决于用户在这台输入输出设备上的追求目标。如果把设备安装的便利性放在第一位,那么建议考虑使用该类设备的通用驱动程序。如果优先考虑设备的运行效率,那么应该使用专门为这台设备编写的驱动程序。

3. 与设备无关的系统软件

除了一些 I/O 软件与设备相关之外,大部分软件是与设备无关的。至于设备驱动程序与设备无关的软件之间的界限如何划分,则随操作系统的不同而不同。具体划分原则取决于系统的设计者怎样权衡系统与设备的独立性、驱动程序的运行效率等诸多因素。对于一些按照设备独立方式实现的功能,出于效率和其他方面的考虑,也可以由设备驱动程序实现。图 3-25 给出了常见的设备无关软件层实现的一些功能。

设备驱动程序的统一接口
设备命名
设备保护
提供一个与设备无关的逻辑块
缓冲
存储设备的块分配
独占设备的分配和释放
错误处理

图 3-25 与设备无关 I/O 软件的功能

一般而言,所有设备都需要的 I/O 功能可以在设备独立的软件中实现。这类软件面向应用层并提供一个统一的接口。

例如,在 UNIX 系统中,像/dev/tty00 这样的设备名,惟一地确定了一个特殊文件的 i 节点,这个 i 节点包含主设备号和从设备号。主设备号用于寻找对应的设备驱动程序,而从设备号提供有关的参数,用来确定要读写的具体设备。

4. 用户层 I/O 软件

通常大部分 I/O 软件都包含在操作系统中,但是用户程序仍有一小部分是与库函数连接在一起的,甚至还有在内核之外运行的程序。通常的系统调用,包括 I/O 系统调用,

是由库函数实现的。例如，一个用 C 语言编写的程序可含有如下的系统调用：

```
count = write(fd, buffer, nbytes);
```

在程序运行期间，该程序将与库函数 write 连接在一起，并包含在运行时的二进制程序代码中。显然，所有这些库函数是 I/O 设备管理系统的组成部分。通常这些库函数所做的工作主要是把系统调用时所用的参数放在合适的位置，由其他的 I/O 过程去实现真正的操作。在这里，输入输出的格式是由库函数完成的。标准的 I/O 库包含许多涉及 I/O 的过程，它们都是作为用户程序的一部分运行的。

例如，C 语言中的 printf 以一个格式串和一些可能的变量作为输入，构造一个 ASCII 字符串，然后调用 write 系统调用输出这个串。

但是，并非所有的用户层 I/O 软件都是由库函数组成的。spooling(假脱机)系统是另一种重要的实现方法。spooling 系统是多道程序设计系统中模拟独占 I/O 设备完成假脱机的一种 I/O 技术。假设有一种典型的假脱机设备——行式打印机，一个进程打开了它，然后很长时间不使用，这样就会导致其他进程无法使用这台打印机打印。

解决方法是创建一个特殊进程，称为守护(daemon)进程，以及一个特殊目录，称为 spooling 目录。当进程要打印一个文件时，首先要生成打印的整个文件，将其放在 spooling 目录下。然后由守护进程完成该目录下文件的打印工作，该进程是惟一个拥有使用打印机特殊文件权限的进程。而且，通过保护特殊文件，防止用户直接使用，可以解决进程空占打印机的问题。

需要指出的是，spooling 技术不仅仅只适用于打印机这类输入输出设备，还可应用到其他一些情况。图 3-26 总结了 I/O 软件的所有层次及每一层的主要功能。



图 3-26 I/O 系统的层次结构与每层的主要功能

图中的箭头给出了 I/O 部分的控制流。这里我们举一个读硬盘文件的例子。当用户程序试图读一个硬盘文件时，需要通过操作系统实现这一操作。与设备无关软件检查高速缓存中是否有要读的数据块。若没有，则调用设备驱动程序，向 I/O 硬件发出一个请求。然后，用户进程阻塞并等待磁盘操作的完成。当磁盘操作完成时，硬件产生一个中

断,转入中断处理程序。中断处理程序检查中断的原因,认识到这是磁盘读取操作已经完成,于是唤醒用户进程取回从磁盘读取的信息,从而结束此次 I/O 请求。用户进程在得到所需的硬盘文件内容之后,继续运行。

3.4.3 通道、DMA 与缓冲技术

1. 通道

引入通道的目的是使数据的传输独立于 CPU,使 CPU 从繁琐的 I/O 工作中解脱出来。设置通道后,CPU 只须向通道发出 I/O 命令。通道收到命令后,从主存中取出本次 I/O 要执行的通道程序,并执行,仅当通道完成了 I/O 任务后,才向 CPU 发出中断信号。

根据信息交换方式的不同可以将通道分为 3 类。

(1) 字节多路通道(byte multiplexor channel)通常都含有许多非分配型子通道,每一个子通道连接一台 I/O 设备。主通道采用时间片轮转法,轮流地为各个子通道服务。只要字节多路通道扫描子通道的速率足够快,而连接到子通道的设备速率不太高时,便不会丢失信息。因此这些子通道连接的是慢速外围设备,如纸带输入机、纸带输出机、卡片输入机、卡片输入机、行式打印机等设备。

(2) 数组选择通道(block selector channel)由于字节多路通道不适于连接高速设备,所以引入数组选择通道。这种通道的传输速率高,可以连接多台高速设备,但由于该通道仅含有一个可分配型通道,因此,在某一段时间内只能执行一个通道程序,为一台设备的输入输出提供服务。这样,一旦某设备占用了通道,将一直由它独占,即使无数据传输而闲置,直到该设备自动释放通道。可见这种通道的利用率很低。

(3) 数组多路通道(block multiplexor channel)结合了数组选择通道传输速率高和字节多路通道能使各个子通道分时并行操作的优点。该通道中含有多个非分配型子通道,因而该通道既具有很高的数据传输速率,又能获得令人满意的通道利用率。其数据传输是按数组方式进行的。

由于通道价格昂贵,导致计算机系统通道数是有限的,这往往会成为输入输出的“瓶颈”问题。在一个单通路的 I/O 系统中主存和设备之间只有一条通路。一旦某通道被设备占用,即使另一通道空闲,连接该通道的其他设备只有等待。解决“瓶颈”问题最有效的方法是增加设备到主机之间的通路,使得主存和设备之间有两条以上的通路。如图 3-27 所示。

2. DMA (direct memory access)技术

直接主存存取(direct memory access,DMA)是指数据在主存与 I/O 设备间的直接成块传送,即主存与 I/O 设备间传送数据块的过程中,不需要 CPU 作任何干涉,只需在过程开始启动(即向设备发出“传送一块数据”的命令)与过程结束(CPU 通过轮询或中断



图 3-27 多通路的 I/O 系统

得知过程是否结束和下次操作是否准备就绪)时由 CPU 进行处理,实际操作由 DMA 硬件直接完成,CPU 在传送过程中可做别的事情。例如,在非 DMA 时,打印 2048 字节,至少需要执行 2048 次输出指令,加上 2048 次中断处理的代价。而在 DMA 情况下,若一次 DMA 可传送 512 字节,则只需要执行 4 次输出指令和处理 4 次打印机中断。若一次 DMA 可传送的字节数大于等于 2048,则只需要执行一次输出指令和处理一次打印机中断。

3. 缓冲技术

缓冲技术可提高外设利用率,使外设尽可能处于忙状态。缓冲技术可以采用硬件缓冲和软件缓冲。硬件缓冲利用专门的硬件寄存器作为缓冲,软件缓冲是通过操作系统来管理的。引入缓冲的主要原因有以下几个方面:

- (1) 缓和 CPU 与 I/O 设备间速度不匹配的矛盾;
- (2) 减少 CPU 的中断频率,放宽对中断响应时间的限制;
- (3) 提高 CPU 和 I/O 设备之间的并行性。

在所有的 I/O 设备与处理机(主存)之间,都使用缓冲区来交换数据。所以操作系统必须组织和管理好这些缓冲区。缓冲可分为单缓冲、双缓冲、多缓冲和环形缓冲。

3.4.4 spooling 技术

spooling 是外围设备联机操作(simultaneous peripheral operations on line)的缩写,常简称为 spooling 系统或假脱机系统。所谓 spooling 技术实际上是用一类物理设备模拟另一类物理设备的技术,是使独占使用的设备变成多台虚拟设备的一种技术,也是一种速度匹配技术。spooling 系统是由“预输入程序”、“缓输出程序”、“井管理程序”以及输入和输出井组成的。其中,输入井和输出井是为了存放从输入设备输入的信息以及作业执行的结果,系统在辅助存储器上开辟的存储区域。spooling 系统的组成和结构如图 3-28

所示。

spooling 系统的工作过程：操作系统初启后激活 spooling 预输入程序，使它处于捕获输入请求的状态。一旦有输入请求消息，spooling 输入程序立即执行，把装在输入设备上的作业输入硬盘的输入井中。并填写好作业表以便在作业执行中要求输入信息时，可以随时找到它们的存放位置。当作业需要输出数据时，可以先将数据送到输出井。当输出设备空闲时，由 spooling 输出程序把硬盘上输出井的数据送到慢速的输出设备上。

spooling 系统中拥有一张作业表，用来登记进入系统的所有作业的作业名、状态和预输入表的位置等信息。每个用户作业拥有一张预输入表，用来登记该作业各个文件的情况，包括设备类型、信息长度及存放位置等。输入井中的作业有 4 种状态：

- 提交状态：作业的信息正从输入设备上预输入。
- 后备状态：作业预输入结束但未被选中执行。
- 执行状态：作业已被选中运行。在运行过程中，它可从输入井中读取数据信息，也可向输出井写信息。
- 完成状态：作业已经撤离，该作业的执行结果等待缓输出。

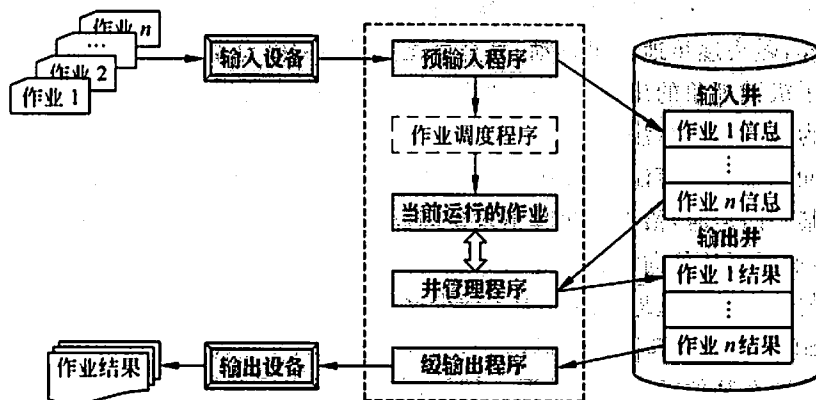


图 3-28 spooling 系统的组成和结构

3.4.5 磁盘调度

磁盘是可供多个进程共享的设备。当多个进程都请求访问磁盘时，为了保证信息的安全，系统每一时刻只允许一个进程启动磁盘进行 I/O 操作，其余的进程只能等待。因此，操作系统应采用一种适当的调度算法，使各进程对磁盘的平均访问（主要是寻道）时间最小。磁盘调度分为移臂调度和旋转调度两类。并且是先进行移臂调度，然后再进行旋转调度。由于访问磁盘最耗时的是寻道时间，因此，磁盘调度的目标应使磁盘的平均寻道

时间最少。

1. 磁盘驱动调度

常用的磁盘调度算法如下：

- 先来先服务 (first-come first-served, FCFS)。这是最简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单,且每个进程的请求都能依次得到处理,不会出现某进程的请求长期得不到满足的情况。但此算法由于未对寻道进行优化,致使平均寻道时间可能较长。
- 最短寻道时间优先 (shortest seek time First, SSTF)。该算法选择这样的进程,其要求访问的磁道与当前磁头所在的磁道距离最近,使得每次的寻道时间最短,但这种调度算法却不能保证平均寻道时间最短。
- 扫描算法 (SCAN)。扫描算法不仅考虑到欲访问的磁道与当前磁道的距离,更优先考虑的是磁头当前的移动方向。例如,当磁头正在由里向外移动时,SCAN 算法所选择的下一个访问对象应使欲访问的磁道既在当前磁道之外,又是距离最近的。这样由里向外地访问,直至再无更外的磁道需要访问时,才将磁臂换向,由外向里移动。这时,同样也是每次选择在当前磁道之内,且距离最近的进程来调度。这样,磁头逐步地向里移动,直至再无更里面的磁道需要访问。显然,这种方式避免了饥饿现象的出现。在这种算法中,磁头移动的规律颇似电梯的运行,故又常称为电梯调度算法。
- 单向扫描调度算法 (CSCAN)。SCAN 存在这样的问题:当磁头刚从里向外移动过某一磁道时,恰有一进程请求访问此磁道,这时该进程必须等待,待磁头从里向外,然后再从外向里扫描完所有要访问的磁道后,才处理该进程的请求,致使该进程的请求被严重地推迟。为了减少这种延迟,CSCAN 算法规定磁头作单向移动。
- N-Step-SCAN 算法,在 SSTF、SCAN 及 CSCAN 几种调度算法中,都可能出现磁臂停留在某位置不动的情况。例如,有一个或几个进程对某一磁道有着较高的访问频率,即反复请求对某一磁道进行 I/O 访问,从而垄断了整个磁盘设备。在高密度盘上更容易出现这种情况,我们把这一现象称为磁臂“粘着” (Armstickiness)。
- FSCAN 算法实质上是 N 步 SCAN 算法的简化。它只将磁盘请求访问队列分成两个子队列。一是当前所有请求磁盘 I/O 的进程形成的队列,由磁盘调度按 SCAN 算法进行处理。另一个队列则是在扫描期间新出现的所有请求磁盘 I/O 进程的队列,把它们插入另一个等待处理的请求队列。这样,所有的新请求都将被推迟到下一次扫描时处理。

2. 旋转调度算法

当移动臂定位后,有多个进程等待访问该柱面时,应如何决定这些进程的访问顺序?这就是旋转调度要考虑的问题。显然系统应该选择延迟时间最短的进程对磁盘的扇区进行访问。当有若干等待进程请求访问磁盘上的信息时,旋转调度应考虑如下情况:

(1) 进程请求访问的是同一磁道上的不同编号的扇区;

(2) 进程请求访问的是不同磁道上的不同编号的扇区;

(3) 进程请求访问的是不同磁道上具有相同编号的扇区。

对于(1)与(2),旋转调度总是让首先到达读写磁头位置下的扇区先进行传送操作;对于(3)的情况,旋转调度可以任选一个读写磁头位置下的扇区进行传送操作。

3.5 文件管理

随着计算机应用需求的不断增长,快速、高效地处理大量的信息是计算机的首要任务之一,而这些信息通常均存储在大容量的外存储器上。但在早期,用户要访问外存储器上的信息是很麻烦的,不仅要考虑信息在外存储器上的存放位置,而且要记住信息在外存储器的分布情况,以便构造 IO 程序。稍不注意,就会破坏已存放的信息。特别是多道程序技术出现后,多个用户之间根本无法预料各个不同程序间的信息在外存储器上是如何分配的。鉴于这些原因,引入文件系统专门负责管理外存储器上信息,使用户可以“按名”高效、快速和方便地存储信息。

3.5.1 文件与文件系统

1. 文件

文件(file)是具有符号名的、在逻辑上具有完整意义的一组相关信息项的集合。例如,一个源程序、一个目标程序、编译程序、一批待加工的数据以及各种文档等都可以各自组成一个文件。

信息项是构成文件内容的基本单位,可以是一个字符,也可以是一个记录。记录可以等长,也可以不等长。一个文件包括文件体和文件说明。文件体是文件的实际内容。文件的说明是操作系统为了管理文件而用的信息,包括文件名、文件内部标识、文件的类型、文件存储地址、文件的长度、访问权限、建立时间以及访问时间等。

文件是一种抽象机制,它隐蔽了硬件和实现细节,提供了将信息保存在磁盘上而且便于以后读取的手段,使用户不必了解信息存储的方法、位置以及存储设备的实际运作方式便可存取信息。因此,文件管理中的一个非常关键的问题在于文件的命名。文件名是在进程创建文件时确定的,以后这个文件将独立于进程存在直到它被显式删除。其他进程

要使用文件时必须显式指出该文件名,操作系统根据文件名对其进行控制和管理。不同操作系统的文件命名规则有所不同,即文件名字的格式和长度因系统而异。

2. 文件系统

由于计算机系统处理的信息量越来越大,所以不可能将所有的信息保存到主存中。特别是在多用户系统中,既要保证各用户文件存放的位置不冲突,又要防止任一用户对外存储器(简称外存)空间占而不用。既要保证各用户文件在未经许可的情况下被窃取、破坏,又要允许在特定的条件下多个用户共享某些文件。因此,需要设立一个公共的信息管理机制来负责统一管理外存和外存上的文件。

所谓文件管理系统就是操作系统中实现文件统一管理的一组软件和相关数据的集合,专门负责管理和存取文件信息的软件机构,简称文件系统。文件系统的功能包括按名存取(即用户可以“按名存取”,而不是“按地址存取”),统一的用户接口(在不同设备上提供同样的接口,方便用户操作和编程),并发访问和控制(在多道程序系统中支持对文件的并发访问和控制),安全性控制(在多用户系统中的不同用户对同一文件可有不同的访问权限),优化性能(采用相关技术提高系统对文件的存储效率、检索和读写性能),以及差错恢复(能够验证文件的正确性,并具有一定的差错恢复能力)。

3. 文件的类型

- (1) 按文件性质和用途可将文件分为系统文件、库文件和用户文件。
- (2) 按信息保存期限分类可将文件分为临时文件、档案文件和永久文件。
- (3) 按文件的保护方式分类可将文件分为只读文件、读写文件、可执行文件和不保护文件。
- (4) UNIX 系统将文件分为普通文件、目录文件和设备文件(特殊文件)。
- (5) 目前常用的文件系统类型有 FAT、Vfat、NTFS、Ext2 和 HPFS 等。

文件分类的目的是对不同文件进行管理,提高系统效率,提高用户界面友好性。当然,根据文件的存取方法和物理结构不同还可以将文件分为不同的类型,将在文件的逻辑结构和文件的物理结构中介绍。

3.5.2 文件的结构和组织

文件的结构是指文件的组织形式。从用户角度看到的文件组织形式称为文件的逻辑结构。文件系统的用户只要知道所需文件的文件名,就可存取文件中的信息,无须知道这些文件究竟存放在什么地方。从实现的角度看文件在文件存储器上的存放方式称为文件的物理结构。

1. 文件的逻辑结构

文件的逻辑结构可分为两大类:一是有结构的记录式文件,它是由一个以上的记录

构成的文件,故又称为记录式文件;二是无结构的流式文件,它是由一串顺序字符流构成的文件。

1) 有结构的记录式文件

在记录式文件中,所有的记录通常都是描述一个实体集的,有着相同或不同数目的数据项,记录的长度可分为定长和不定长两类。

(1) 定长记录:它是指文件中所有记录的长度都是相同的。所有记录中的各个数据项,都处在记录中相同的位置,具有相同的顺序及相同的长度,文件的长度用记录数目表示。定长记录的特点是处理方便,开销小,是目前较常用的一种记录格式,被广泛用于数据处理中。

(2) 变长记录:它是指文件中各记录的长度不同。这是因为:

- 一个记录中所包含的数据项数目可能不同。如书的著作者和论文中的关键词;
- 数据项本身的长度不定。例如,病历记录中的病因和病史,科技情报记录中的摘要等。但不论哪一种,在处理前每个记录的长度是可知的。

2) 无结构的流式文件

文件体为字节流,不划分记录。无结构的流式文件通常采用顺序访问方式,并且,每次读写访问可以指定任意数据长度,其长度以字节为单位。对流式文件访问时,应利用读写指针指出下一个要访问的字符。可以把流式文件看作记录式文件的一个特例。在UNIX系统中,所有的文件都被看作流式文件。即使有结构的文件,也被视为流式文件,系统不对文件进行格式处理。

2. 文件的物理结构

文件的物理结构是指文件的内部组织形式,即文件在物理存储设备上的存放方法。由于文件的物理结构决定了文件信息在文件存储设备上的存放位置,所以文件的逻辑块号到物理块号的转换也是由文件的物理结构决定的。根据用户和系统管理上的需要,可采用多种方法来组织文件,下面介绍几种常见的文件物理结构。

1) 连续结构

连续结构也称顺序结构。它将逻辑上连续的文件信息(如记录)依次存放在连续编号的物理块上。只要知道文件的起始物理块号和文件的长度,就可以很方便地进行文件的存取。例如,文件W.TXT占用了50、51、52和53号物理块,系统只需将文件的起始块号50和文件的长度放在文件目录中该文件所对应的文件说明中即可,如图3-29所示。

连续结构的最佳应用场合是对文件的诸记录进行批量存取时,这在所有的逻辑文件中,其存取效率是最高的。但在交互应用的场合,如果用户(程序)要求随机地查找或修改单个记录,此时系统需要逐个地查找诸记录,采用连续结构所表现出来的性能就可能很差,尤其是当文件较大时情况将更为严重。连续结构的另一个缺点是,不便于记录的增加

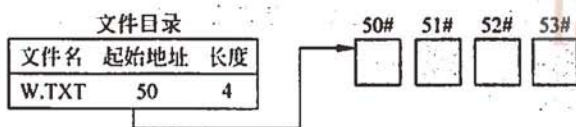


图 3-29 连续结构

或删除操作。

2) 链接结构

链接结构也称串联结构,它是将逻辑上连续的文件信息(如记录)存放在不连续的物理块上,每个物理块设有一个指针指向下一个物理块。因此,只要知道文件的第一个物理块号,就可以按链指针查找整个文件。

例如,文件 W.TXT 占用了 60、86、92 和 103 号物理块,文件的起始块号 60 放在文件说明中,如图 3-30 所示。

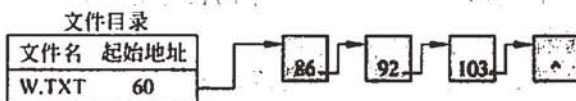


图 3-30 链接结构

3) 索引结构

采用索引结构将逻辑上连续的文件信息(如记录)存放在不连续的物理块中。系统为每个文件建立一张索引表,索引表记录了文件信息所在的逻辑块号对应的物理块号,并将索引表的起始地址放在文件对应的文件目录项中。

例如,文件 W.TXT 占用了 60、86、92 和 103 号物理块,文件索引表存放在 98 号物理块中,W.TXT 文件的文件目录项指向文件索引表,如图 3-31 所示。



图 3-31 索引结构

访问 W.TXT 文件的过程:系统按文件名“W.TXT”查文件目录表,根据索引表的起始地址将 98# 索引表块读入主存,按索引表查找对应的物理块号并将物理块读入主存。

4) 多个物理块的索引表

索引表是在文件建立时由系统自动建立的,并与文件一起存放在同一文件卷上。一

个文件的索引表根据文件大小的不同,占用物理块的个数不等,一般占一个或几个物理块。多个物理块的索引表可有两种组织方式:链接文件 and 多重索引方式。

(1) 链接文件方式:将多个索引表块按链接文件的方式串联起来。

例如,每个索引表项占 4 个字节(物理块号的表示范围为 $0 \sim 2^{32}$),若物理块的大小为 512 字节,则一个物理块可存放 127 个索引表项和一个链接字,如图 3-32 所示。

(2) 多重索引方式:具有多个物理块的索引表的另一种有效组织方式是多重索引方式。其结构如图 3-33 所示。

(3) UNIX 文件系统的索引结构。

UNIX 文件系统采用的是三级索引结构。文件系统中的 inode 是基本的构件,它表示文件系统树型结构的节点。UNIX 有直接、一级间接、二级间接和三级间接四种寻址方式。

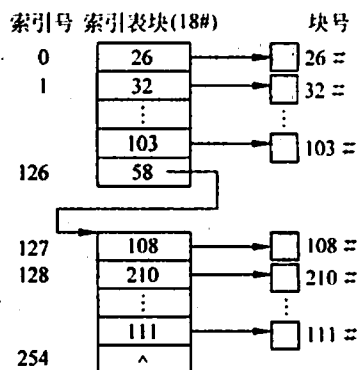


图 3-32 链接方式的索引结构

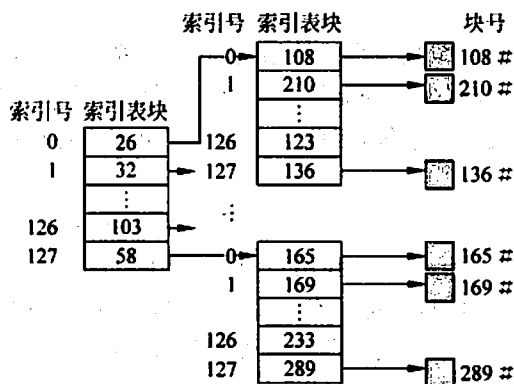


图 3-33 多重索引结构

3.5.3 文件目录

为了实现“按名存取”,系统必须为每个文件设置用于描述和控制文件的数据结构。它至少应包括文件名和存放文件的物理地址,这个数据结构称为文件控制块 FCB。文件控制块的有序集合称为文件目录。换句话说,文件目录是由文件控制块组成的,专门用于文件的检索。文件控制块 FCB 也称为文件说明或文件目录项(简称目录项)。

1. 文件控制块 FCB

文件控制块 FCB 中包含以下 3 类信息：基本信息类、存取控制信息类和使用信息类。

(1) 基本信息类：文件名、文件的物理地址、文件长度和文件块数等。

(2) 存取控制信息类：文件的存取权限。如 UNIX 用户的文件存取权限分为文件主、同组用户和一般用户 3 类，规定了这 3 类用户的读写执行(RWX)权限。

(3) 使用信息类：文件建立日期，最后一次修改日期，最后一次访问日期，以及当前的使用信息，如打开文件的进程数，在文件上的等待队列等。目录文件也是使用信息类文件。文件目录是由文件控制块组成的，专门用于文件的检索。文件目录可以存放在文件存储器的固定位置，也可以以文件形式存放在磁盘上，我们将这种特殊的文件称之为目录文件。

2. 目录结构

文件目录结构的组织方式直接影响到文件的存取速度，关系到文件的共享性和安全性。因此组织好文件的目录是设计文件系统的重要环节。常见的目录结构有 3 种：一级目录结构、二级目录结构和多级目录结构。

1) 一级目录结构

一级目录的整个目录组织是一个线性结构，在整个系统中只需建立一张目录表，系统为每个文件分配一个目录项(文件控制块)。一级目录结构简单，但缺点是查找速度慢，不允许重名和不便于实现文件共享等，因此它主要用在单用户环境中。

2) 二级目录结构

为了克服一级目录结构存在的缺点，引入了二级目录结构。二级目录结构是由主文件目录 MFD(Master File Directory)和用户目录 UFD(User File Directory)组成的。在主文件目录中，每个用户文件目录都占有一个目录项，其目录项中包括用户名和指向该用户目录文件的指针。用户目录由用户所有文件的目录项组成。如图 3-34 所示。

二级目录结构基本上克服了单级目录的缺点。其优点如下：提高了检索目录的速度，较好地解决了重名问题。采用二级目录结构也存在一些问题。该结构虽然能有效地将多个用户隔离开，这种隔离在各个用户之间完全无关时是一个优点，但当多个用户之间需要相互合作共同完成一个大任务时，且一用户又需要访问其他用户的文件时，这种隔离便成为一个缺点，因为这种隔离使诸用户之间不便于共享文件。

3) 多级目录结构

为了解决以上问题，多道程序设计系统中常采用多级目录结构。这种目录结构像一棵倒置的有根树，所以也称为树形目录结构。从树根向下，每一个节点是一个目录，叶节点是文件。MS-DOS 和 UNIX 等操作系统均采用多级目录结构。

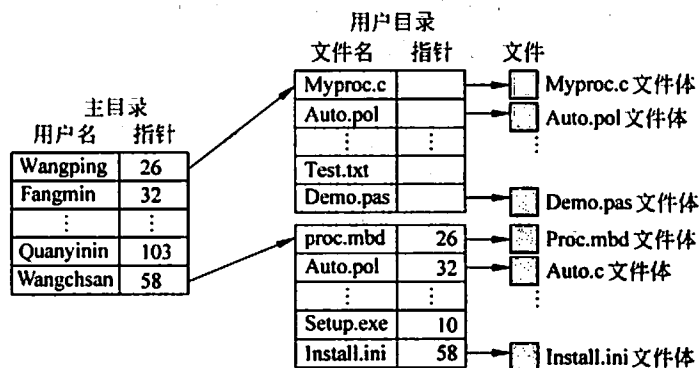


图 3-34 二级目录结构

在采用多级目录结构的文件系统中,用户访问一个文件时必须指出文件所在的路径名,路径名是由从根目录开始到该文件的通路上所有各级目录名拼接起来的。各目录名之间,目录名与文件名之间需要用分隔符隔开。例如,MS-DOS 中的分隔符为“\”,UNIX 中的分隔符为“/”。绝对路径名(absolute path name)是指从根目录“/”开始的完整文件名,即它是由从根目录开始的所有目录名以及文件名构成的。

在多级目录中存取一个文件需要用文件全名,这就意味着允许用户在自己的目录中使用与其他用户文件相同的文件名,由于各用户使用不同的目录,虽二者使用了相同的文件名,但它们的文件全名仍不相同,这就解决了重名问题。采用多级目录结构提高了检索目录的速度:例如,采用单级目录查找一个文件最多需查遍系统目录文件中的所有文件目录项,平均也要查一半文件目录项。而多级目录查找一个文件最多只要查遍文件路径上根目录文件和子目录文件中的目录项。

3.5.4 存取方法和存储空间的管理

1. 文件的存取方法

文件的存取方法是指读写文件存储器上的一个物理块的方法。通常有顺序存取和随机存取。顺序存取是指对文件中的信息按顺序依次读写。随机存取是指对文件中的信息可以按任意的次序随机地读写文件中的信息。

(1) 顺序存取法:在提供记录式文件结构的系统中,顺序存取法就是严格按物理记录排列的顺序依次读取。如果当前读取的是 R_i 记录,下一次要读取的记录自动地确定为 R_{i+1} 。在只提供无结构的流式文件中,顺序存取法是按读写的偏移(offset)从当前位置开始读写,每读完一段信息,读写偏移自动加上这段信息的长度,以便读下一段信息。下面着重讨论记录式文件的存取法。

(2) 直接存取法: 直接存取法允许用户随意存取文件中任意一个物理记录。对于无结构的流式文件, 采用直接存取法时, 必须事先将读写偏移移动到待读写信息的位置上, 然后再进行读写。

(3) 按键存取法: 按键存取法是直接存取法的一种, 它不是根据记录的编号或地址来存取文件中的记录, 而是根据文件中各记录的某个数据项内容来存取记录的, 这种数据项称之为“键”。因此, 将这种存取法称之为按键存取法。

2. 文件存储空间的管理

外存具有大容量的存储空间, 供多用户共享, 用户执行程序经常要在磁盘上存储文件和删除文件。因此, 文件系统必须对磁盘空间进行管理。外存空闲空间管理的数据结构通常称为磁盘分配表(disk allocation table)。常用的空闲空间的管理方法有位示图、空闲块表和空闲块链三种。

1) 空闲区表

将外存空间上一个连续未分配区域称为“空闲区”。操作系统为磁盘外存上所有空闲区建立一张空闲表, 每个表项对应一个空闲区, 空闲表中包含序号、空闲区的第一块号以及空闲块的块数等信息。它适用于连续文件结构。

2) 位示图

这种方法是在外存上建立一张位示图(bitmap), 记录文件存储器的使用情况。每一位对应文件存储器上的一个物理块, 值 0 和 1 分别表示空闲和占用。文件存储器上的物理块依次编号为: 0, 1, 2, ...。假如系统中字长为 32 位, 那么位示图中的第一个字对应文件存储器上的 0, 1, 2, ..., 31 号物理块; 第二个字对应文件存储器上的 32, 33, 34, ..., 63 号物理块; 以下类推。

这种方法的主要特点是位示图的大小由磁盘空间的大小(物理块总数)决定, 位示图的描述能力强, 适合各种物理结构。

3) 空闲块链

每个空闲物理块中有指向下一个空闲物理块的指针, 所有空闲物理块构成一个链表, 链表的头指针放在文件存储器的特定位置上(如管理块中)。不需要磁盘分配表, 节省空间。每次申请空闲物理块只须根据链表的头指针取出第一个空闲物理块, 根据第一个空闲物理块的指针可找到第二个空闲物理块, 依次类推即可。

4) 成组链接法

在 UNIX 系统中, 将空闲块分成若干组, 每 100 个空闲块为一组, 每组的第一个空闲块登记下一组空闲块的物理盘块号和空闲块总数, 假如一个组的第一个空闲块号等于 0, 意味着该组是最后一组, 即无下一组空闲块。

3.5.5 文件的使用

文件系统将用户的逻辑文件按一定的组织方式转换成物理文件存放到文件存储器上,也就是说文件系统为每个文件与该文件在磁盘上的存放位置建立了对应关系。当用户使用文件时,文件系统通过用户给出的文件名,查出对应文件的存放位置,读出文件的内容。在多用户环境下,为了保护文件安全起见,操作系统为每个文件建立和维护关于文件主、访问权限等方面的信息。为此操作系统在操作级(命令级)和编程级(系统调用和函数)向用户提供文件服务。

操作系统在操作级向用户提供的命令有目录管理类命令、文件操作类命令(如复制、删除和修改)以及文件管理类命令(如设置文件权限)等。

操作系统在编程级向用户提供的文件操作系统调用列举如下。

- 创建文件: 如 `create(文件名, 参数表)`。
- 撤销文件: 如 `delete(文件名)`。
- 打开文件: 如 `open(文件名, 参数表)`。
- 关闭文件: 如 `close(文件名)`。
- 读文件: 如 `read(文件名, 参数表)`。
- 写文件: 如 `write(文件名, 参数表)`。

3.5.6 文件的共享和保护

1. 文件的共享

文件共享是指不同用户进程使用同一文件。这不仅是不同用户完成同一任务所必须的功能,而且还可以节省大量的主存空间,减少由于文件复制而增加的外存访问次数。文件共享有多种形式,采用文件名和文件说明分离的目录结构有利于实现文件共享。

常见的文件链接有硬链接和符号链接两种。

1) 硬链接

文件的硬链接是指两个文件目录表目指向同一个索引节点的链接,该链接也称基于索引节点的链接。换句话说,硬链接是指不同文件名与同一个文件实体的链接。文件硬链接不利于文件主删除他们拥有的文件,因为文件主删除他们拥有的共享文件时,必须首先删除(关闭)所有的硬链接,否则就会造成共享该文件的用户的目录表目指针悬空。

例如,UNIX 系统中的“`ln`”命令可以将多个文件名与一个文件体建立链接,其格式为:

`ln 文件名 新文件名` 或 `ln 文件名 目录名`

ls 命令放在/bin 子目录下,我们可在/usr/bin 子目录下设置一个 DOS 兼容的命令 dir,该命令实为执行 ls 命令。使用命令“ln”可给一个已存在文件增加一个新文件名,即文件链接数增加 1。此种链接是不能跨越文件系统的。为了共享文件,只是在两个不同子目录下取了不同的文件名 ls 和 dir,但它们具有相同的索引节点。UNIX 这种文件的结构称为树形带勾链的目录结构。在文件的索引节点中,di_nlink 变量表示链接到该索引节点上的链接数。在用命令“ls -l”长列表显示时,文件的第 2 个数据项表示链接数。

2) 符号链接

符号链接建立的新文件或目录是原文件或目录路径名的映射。当访问一个符号链接时,系统通过该映射找到原文件的路径,并对其进行访问。

例如,UNIX 系统中的“ln -s”命令建立符号链接时,系统为共享的用户创建一个 link 类型的新文件,将这新文件登录在该用户共享目录项中,这个 link 型文件包含链接文件的路径名。该类文件在用 ls 命令长列表显示时,文件链接数为 1。

采用符号连接可以跨越文件系统,甚至可以通过计算机网络连接到世界上任何地方的机器中的文件,此时只须提供该机器所在的地址,以及在该机器中的文件路径。

符号连接的缺点:其他用户读取符号连接的共享文件比读取硬连接的共享文件需要增加读盘操作的次数。因为其他用户读符号连接的共享文件时,系统根据给定的文件路径名,逐级找目录,通过多次读盘操作才能找到该文件的索引节点。而采用硬连接的共享文件的目录文件表目中已包括了共享文件的索引节点号。

2. 文件的保护

文件系统对文件的保护常采用存取控制方式。所谓存取控制就是不同的用户对文件的访问有不同的权限,以防止文件被未经文件主同意的用户访问。

1) 存取控制矩阵

理论上存取控制方法可用存取控制矩阵,它是一个二维矩阵,一维列出计算机的全部用户,另一维列出系统中的全部文件,矩阵中每个元素 A_{ij} 表示第 i 个用户对第 j 个文件的存取权限。通常存取权限有可读、可写、可执行以及它们的组合,如表 3-3 所示。

表 3-3 存取控制矩阵

文件 用户	ALPHA	BETA	REPORT	SQRT	...	
张军	RWX	...	R-X	...		
李晓钢	R-X	...	RWX	R-X	...	
王伟	...	RWX	R-X	R-X		
赵凌	RWX		
...	...					

存取控制矩阵在概念上是简单清楚的,但实现上却有困难。当一个系统用户数和文件数很大时,二维矩阵要占很大的存储空间,验证过程也将耗费许多系统时间。

2) 存取控制表

存取控制矩阵由于太大而往往无法实现。一个改进的办法是按用户对文件的访问权力的差别对用户进行分类。由于某一文件往往只与少数几个用户有关,所以这种分类方法可使存取控制表大为简化。UNIX 系统就是使用这种存取控制表方法的。它把用户分成三类:文件主、同组用户和其他用户,每类用户的存取权限为可读、可写、可执行以及它们的组合。在用 ls 长列表显示时,每组存取权限用三个字母 RWX 表示,如读、写和执行中那一样存取不允许则用“-”字符表示。用 ls -l 长列表显示 ls 文件如下:

```
-r-xr-xr-t 1 bin bin 43296 May 13 1997 /opt/K/SCO/Unix/5.0.4Eb/bin/ls
```

显示信息前 2-10 共 9 个字符表示文件的存取权限。每 3 个字符为一组,分别表示文件主、同组用户和其他用户的存取权限。由于存取控制表对每个文件按用户分类,所以该存取控制表可存放在每个文件的文件控制块中。UNIX 只需 9 位二进制数表示三类用户对文件的存取权限,该权限存在文件索引节点的 di_mode 中。

3) 用户权限表

改进存取控制矩阵的另一种方法是以用户或用户组为单位将用户可存取的文件集中起来存入表中。这个表称为用户权限表,表中每个表目表示该用户对相应文件的存取权限,这相当于把存取控制矩阵简化为一行。

4) 密码

在创建文件时,由用户提供一个密码,在文件存入磁盘时用该密码对文件内容加密。执行读取操作时,要对文件进行解密,只有知道密码的用户才能读取文件。

3.5.7 系统的安全与可靠性

1. 系统的安全

系统的安全涉及两类不同的问题,一类涉及技术、管理、法律、道德和政治等问题,另一类涉及操作系统的安全机制。随着计算机应用范围的扩大,在所有稍具规模的系统中,都从多个级别上来保证系统的安全性。一般从 4 个级别上对文件进行安全性管理:系统级、用户级、目录级和文件级。

(1) 系统级安全管理的主要任务是不允许未经授权的用户进入系统,从而也防止了他人非法使用系统中各类资源(包括文件)。系统级管理的主要措施有:注册与登录。

(2) 用户级安全管理是通过对所有用户分类和对指定用户分配访问权。不同的用户对不同文件具有不同的存取权限。例如,UNIX 系统将用户分为文件主、组用户和其他用

户。有的系统将用户分为超级用户、系统操作员和一般用户。

(3) 目录级安全管理是为了保护系统中各种目录而设计的,它与用户权限无关。为保证目录的安全,规定只有系统核心才具有写目录的权利。

(4) 文件级安全管理是通过系统管理员或文件主对文件属性的设置来控制用户对文件的访问。通常可设置以下几种属性:只执行、隐含、只读、读写、共享和系统。用户对文件的访问将由用户访问权、目录访问权限及文件属性三者的权限确定。或者说是有效权限和文件属性的交集。如只读文件,尽管用户的有效权限是读/写,但都不能对只读文件进行修改、更名和删除。对于一个非共享文件,将禁止在同一时间内由多个用户对它们进行访问。通过上述4级文件保护措施,可有效地实现对文件的保护。

2. 文件系统的可靠性

文件系统的可靠性是指系统抵抗和预防各种物理性破坏和人为破坏的能力。比起计算机的损坏,文件系统破坏往往后果更加严重。例如,将开水撒在键盘上引起的故障,尽管伤脑筋但毕竟可以修复。但如果文件系统破坏了,在很多情况下是无法恢复的。特别是对于哪些程序文件、客户档案、市场计划或其他数据文件丢失的客户来说,这不亚于一场大的灾难。尽管文件系统无法防止设备和存储介质的物理损坏,但至少应能保护信息。

1) 转储和恢复

文件系统中无论硬件或软件都会发生损坏和错误。例如自然界的闪电、电压的突变、火灾和水灾等均可能引起软、硬件的破坏。为了使文件系统万无一失,应当采用相应的措施。最简单和常用的措施是通过转储操作,形成文件或文件系统的多个副本。这样一旦系统出现故障,利用转储的数据使得系统恢复成为可能。常用的转储方法有:静态转储和动态转储、海量转储和增量转储。

2) 日志文件

在计算机系统的工作过程中,操作系统把用户对文件的插入、删除和修改操作写入日志文件。一旦发生故障,操作系统恢复子系统利用日志文件来进行系统故障恢复,并可协助后备副本进行介质故障恢复。

3) 文件系统的一致性

影响文件系统可靠性的因素之一是文件系统的一致性问题。很多文件系统是先读取磁盘块到主存,在主存进行修改,修改完毕再写回磁盘。但如读取某磁盘块,修改后再将信息写回磁盘前系统崩溃,则文件系统就可能会出现不一致性状态。如果这些未被写回的磁盘块是索引节点块、目录块或空闲块,那么后果是不堪设想的。通常的解决方案是采用文件系统的一致性检查。一致性检查包括块的一致性检查和文件的一致性检查。

3.6 作业与作业管理

作业是系统为完成一个用户的计算任务(或一次事务处理)所做的工作总和。例如,对用户编写的源程序,需要经过编译、连接装入以及执行等步骤得到结果。其中的每一个步骤称之为作业步。操作系统中用来控制作业录入、执行和撤销的一组程序称之为作业管理程序。操作系统可以进一步为每个作业创建作业步进程,完成用户的工作。

3.6.1 作业管理

1. 作业控制

用户作业可以采用脱机和联机两种控制方式控制作业运行。在脱机控制方式中,作业运行的过程是无须人工干预的,因此用户必须将自己想让计算机干什么的意图用作业控制语言(JCL)编写成作业说明书,并连同作业一起提交给计算机系统。在联机控制方式中,操作系统为用户提供了一组联机命令,用户可以通过终端键入命令,将自己想让计算机干什么的意图告诉计算机,以控制作业的运行过程。因此整个作业的运行过程是需要人工干预的。

作业由程序、数据和作业说明书 3 部分组成。

作业说明书包括作业基本情况、作业控制和作业资源要求的描述。它体现了用户的控制意图。其中,作业基本情况包括用户名、作业名、编程语言和最大处理时间等。作业控制描述包括作业控制方式、作业步的操作顺序以及作业执行出错处理。作业资源要求描述包括处理时间、优先级、主存空间、外设类型和数量,以及实用程序要求等。

2. 作业状态及转换

作业的状态分为 4 种:提交、后备、执行和完成。

(1) 提交:作业提交给计算机中心,通过输入设备送入计算机系统的过程称之为提交状态。

(2) 后备:作业通过 spooling 系统输入到计算机系统的后备存储器(磁盘)中,随时等待作业调度程序调度时的状态。

(3) 执行:一旦作业被作业调度程序选中,为其分配了必要的资源,并为其建立相应的进程后,该作业便进入了执行状态。

(4) 完成:当作业正常结束或异常终止时,作业进入完成状态。此时有作业调度程序对该作业进行善后处理。如撤销作业的作业控制块,收回作业所占的系统资源,将作业的执行结果形成输出文件放到输出井中,由 spooling 系统控制输出。

作业的状态及转换如图 3-35 所示。

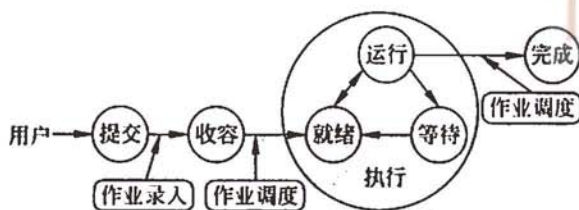


图 3-35 作业的状态及其转换

3. 作业控制块和作业后备队列

所谓作业控制块 JCB 是记录作业各种有关信息的登记表。JCB 是作业存在的惟一标志,其中包括用户名、作业名和状态标志等信息。

由于输入井中有较多的后备作业,为了便于作业调度程序调度,通常将作业控制块排成一个或多个队列,这些队列称之为作业后备队列。也就是说,作业后备队列是由若干个 JCB 组成的。

3.6.2 作业调度

选择调度算法需要考虑的因素:与系统的整体设计目标一致,均衡使用系统资源,以及平衡系统和用户的要求。对用户来说,作业能“立即执行”往往难以做到,但是应保证进入系统的作业在规定的截止时间内完成,而且系统应设法缩短作业的平均周转时间。

1. 作业调度算法

常用的作业调度算法如下所述。

- (1) 先来先服务:按作业到达先后为依据进行调度,即先启动等待时间最长的作业。
- (2) 短作业优先:以要求运行时间长短为依据进行调度,即启动运行时间最短的作业。
- (3) 响应比高优先:响应比高的作业优先启动。

响应比定义如下: $R_p = \frac{\text{作业响应时间}}{\text{作业执行时间}}$,其中作业响应时间为作业进入系统后的等候

时间与作业的执行时间之和,即 $R_p = 1 + \frac{\text{作业等待时间}}{\text{作业执行时间}}$ 。

响应比高者优先算法是在每次调度前都要计算所有被选作业(在作业后备队列中)的响应比,然后选择响应比最高的作业执行。该算法比较复杂,系统开销大。

(4) 优先级调度算法:可由用户指定作业优先级,优先级高的作业先启动。也可由系统根据作业要求的紧迫程度确定,或者照顾“I/O 繁忙”的作业,以便充分发挥外设的效率等。

(5) 均衡调度算法：这种算法的基本思想是根据系统的运行情况和作业本身的特性对作业进行分类。作业调度程序轮流地从这些不同类别的作业中挑选作业执行。这种算法力求均衡地使用系统的各种资源，既注意发挥系统效率，又使用户满意。

2. 作业调度算法性能的衡量指标

在一个以批量处理为主的系统中，通常用平均周转时间或平均带权周转时间来衡量调度性能的优劣。假设作业 $J_i (i=1, 2, \dots, n)$ 的提交时间为 t_s ，执行时间为 t_n ，作业完成时间为 t_a ，则作业 J_i 的周转时间 T_i 和带权周转时间 W_i 分别定义为：

$$T_i = t_a - t_s \quad (i=1, 2, \dots, n), \quad W_i = T_i / t_n \quad (i=1, 2, \dots, n)$$

n 个作业的平均周转时间 T 和平均带权周转时间 W 分别定义为：

$$T = \frac{1}{n} \sum_{i=1}^n T_i, \quad W = \frac{1}{n} \sum_{i=1}^n W_i$$

从用户的角度出发，总是希望自己的作业在提交后能立即执行。这意味着当等待时间为零时作业的周转时间最短，即 $T_i = t_n$ 。但是作业的执行时间 t_n 并不能直观地衡量出系统的性能，而带权周转时间 W_i 却能直观反应系统的调度性能。站在整个系统的立场上，不可能满足每个用户的这种要求，而只能力求系统的平均周转时间或平均带权周转时间最小。

3.6.3 用户界面

用户界面(user interface)是计算机中实现用户与计算机通信的软件和硬件部分的总称。用户界面也称用户接口或人机界面。

用户界面的硬件部分包括用户向计算机输入数据或命令的输入装置，及由计算机输出供用户观察或处理的输出装置。用户界面的软件部分包括用户与计算机相互通信的协议、约定、操纵命令及其处理软件。目前常用的输入输出装置有键盘、鼠标、显示器和打印机等。常用的人机通信方法有命令语言、选项、表格填充及直接操纵等。从计算机用户界面的发展过程来看，可分为若干阶段。

(1) 控制面板式用户界面。

计算机发展早期，用户通过控制台开关、板键或穿孔纸带向计算机送入命令或数据，而计算机通过指示灯及打印机输出运行情况或结果。这种界面的特点是人去适应现在看来十分笨拙的计算机。

(2) 字符用户界面。

字符用户界面是基于字符型设备的。用户通过键盘或其他输入设备输入字符，由显示器或打印机输出字符。字符用户界面的优点是功能强，灵活性好，屏幕开销少。缺点是操作步骤繁琐，学会操作也较费时。

(3) 图形用户界面。

随着文字、图形、声音和图像等多媒体技术的出现,使各种图形用户界面应运而生,用户既可使用传统的字符,也可使用图形、图像和声音同计算机进行交互。操作更为自然,更加方便。形声兼备的多媒体技术进一步推广、发展与完善。现代界面的关键技术在于超文本。超文本的“超”体现在它不仅包括文本,还包括图像、音频和视频等多媒体信息,即将文本的概念扩充到超文本。超文本的最大特点是具有指向性。

(4) 新一代用户界面。

虚拟现实技术将用户界面发展到一个新阶段:人将作为参与者,以自然的方式与计算机生成的虚拟环境进行通信。以用户为中心,自然、高效、高带宽、非精确及无地点限制等是新一代用户界面的特征。多媒体、多通道及智能化是新一代用户界面的技术支持。语音、自然语言、手势、头部跟踪、表情和视线跟踪等新的、更加自然的交互技术将为用户提供更方便的输入技术。计算机将通过多种感知通道来理解用户的意图,实现用户的要求。计算机不仅以二维屏幕向用户输出,而且以真实感(立体视觉、听觉、嗅觉和触觉等)的计算机仿真环境向用户提供真实的体验。

3.7 网络操作系统和嵌入式操作系统基础知识

3.7.1 网络操作系统

计算机网络系统除了硬件,还需要有系统软件,二者结合构成计算机网络的基础平台。操作系统是最重要的系统软件。网络操作系统是网络用户和计算机网络之间的一个接口,它除了应具备通常操作系统应具备的基本功能外,还应有联网功能,支持网络体系结构和各种网络通信协议,提供网络互联功能,支持有效的、安全可靠的数据传送。

一个典型的网络操作系统的特征包括硬件独立性(网络操作系统可以运行在不同的网络硬件上,可以通过网桥或路由器与别的网络连接)、多用户支持(应能同时支持多个用户对网络的访问,应对信息资源提供完全的安全和保护功能)、支持网络实用程序及其管理功能(如系统备份、安全管理、容错和性能控制)、多种客户端支持(如 Windows NT 网络操作系统,及 OS/2、Windows 98 和 UNIX 等多种客户端,极大地方便了网络用户)、提供目录服务(以单一逻辑的方式让用户访问位于世界范围内的所有网络服务和资源的技术)以及支持多种增值服务(如文件服务、打印服务、通信服务和数据库服务)等。

网络操作系统可分为 3 类:

(1) 集中模式:集中式网络操作系统是由分时操作系统加上网络功能演变而来的。系统的基本单元由一台主机和若干台与主机相连的终端构成,将多台主机连接起来形成

了网络,信息的处理和控制是集中的。UNIX 就是这类系统的典型例子。

(2) 客户机/服务器模式:是流行的网络工作模式。这种网络模式可分为:服务器和客户机。服务器是网络的控制中心,其任务是向客户提供一种或多种服务。服务器可有多种类型,如提供文件/打印服务的文件服务器等。客户:这是用于本地处理和访问服务器的站点。客户机中包含本地处理软件和访问服务器上服务程序的软件接口。

(3) 对等模式(peer-to-peer):采用这种模式的操作系统网络,各个站点是对等的。它既可作为客户访问其他站点,又可作为服务器向其他站点提供服务。在网络中既无服务处理中心。也无控制中心。或者说,网络的服务和控制功能分布在各个站点上。可见该模式具有分布处理及分布控制的特征。

现代操作系统已把网络功能包含到操作系统的内核中,作为操作系统核心功能的一个组成部分。微软公司的 Windows NT,AT & T 公司的 UNIX System V、Sun 公司的 SunOS、HP 公司的 HP/OX、IBM 公司的 AIX 以及 Linux 等都已把 TCP/IP 网络功能包含在内核中。

Windows NT 的 I/O 系统包含输入输出管理程序、文件系统、缓冲存储管理系统、设备驱动程序以及网络驱动程序。

网络操作系统是整个网络的灵魂,它决定了网络的功能,并由此决定了不同网络的应用领域及方向。网络操作系统主要有 UNIX、Windows NT 和 NetWare 三大阵营。各种网络操作系统具有不同的特点。随着网络技术的发展,新的网络操作系统还会不断出现,用户可根据自己的需要进行选择,而不要仅局限于其技术水平的高低。

3.7.2 嵌入式操作系统

嵌入式系统中的操作系统称为嵌入式操作系统。嵌入式操作系统是运行在嵌入式智能芯片环境中,对整个智能芯片以及其控制的各种部件和装置等资源进行统一协调、调度、指挥和控制的系统软件。嵌入式系统广泛应用于各种工业控制系统、计算机外设、微波炉、洗衣机和冰箱等低端设备,也用在信息化家电、掌上电脑、机顶盒、WAP 手机和路由器等高端设备中。

1. 嵌入式操作系统的特点

一般而言,嵌入式操作系统不同于一般意义的计算机操作系统,它有占用空间小、执行效率高、便于个性化定制和软件固化存储等特点。嵌入式操作系统和其他嵌入式软件都具有如下特点。

(1) 微型化。由于硬件平台的局限性,如主存少,字长短,运行速度有限,能源少(用微型电池),外部设备和控制对象千变万化。因此,不论从性能还是从成本角度考虑,都不允许它占用很多资源。应在保证应用功能的前提下,以微型化作为特点来设计嵌入式

操作系统的结构与功能。

(2) 可定制。嵌入式操作系统的运行平台多种多样,应用更是五花八门,所以表现出专业化的特点。从减少成本和缩短研发周期考虑,要求它能运行在不同的微处理器平台上,能针对硬件变化进行结构与功能上的配置,以满足不同的应用需要。

(3) 实时性。嵌入式操作系统广泛应用于过程控制、数据采集、传输通信、多媒体信息及关键要害领域需要迅速响应的场合,实时响应要求严格,因此实时性是其主要特点之一。

(4) 可靠性。系统构件、模块和体系结构必须达到应有的可靠性,对关键要害应用还要提供容错和故障防护措施,以进一步提高可靠性。

(5) 易移植性。为了提高系统的可移植性,通常采用硬件抽象层(hardware abstraction level, HAL)和板级支撑包(board support package, BSP)的底层设计技术。HAL 提供了与设备无关的特性,屏蔽硬件平台的细节和差异,向操作系统上层提供统一接口,保证了系统的可移植性。一般由硬件厂家按给定的编程规范提供 BSP,保证嵌入式操作系统可在新推出的微处理器平台上运行。

2. 嵌入式系统开发环境

嵌入式系统开发环境通常配有源代码级可配置的系统模块设计、丰富的同步原语、可选择的调度算法、可选择的主存分配策略、定时器与计数器、多方式中断处理支持、多种异常处理选择、多种通信方式支持、标准 C 语言库、数学运算库和开放式应用程序接口。较著名的嵌入式操作系统有 Windows CE、VxWorks、pSOS、Palm OS 和 μ C/OS-II。

3.8 操作系统实例

3.8.1 UNIX 操作系统

1. UNIX 系统的结构

UNIX 操作系统是由美国贝尔实验室发明的一种多用户、多任务的分时操作系统。现已发展成为当前使用普遍、影响深远的工业界主流操作系统,成为重要的企业级操作平台。UNIX 广泛运行于中型机和小型机等各种环境,用于大型信息系统的关键业务处理,如数据库和 Internet 主机。UNIX 结构如图 3-36 所示。UNIX 最内层硬件提供基本服务,内核提供全部应用程序所需的各种服务。

2. 文件系统

UNIX 文件系统的目录结构是树形带交叉勾连的结构。根目录记为“/”,非叶节点为目录文件,叶节点可以是目录文件、也可以是文件或特殊文件。目录是一个包含目录项的

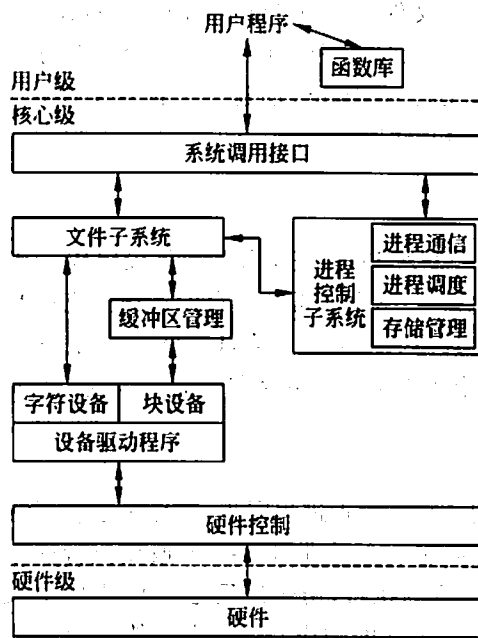


图 3-36 UNIX 系统结构

文件,在逻辑上,可以认为每个目录项都包含一个文件名,同时还包含说明该文件属性的信息。文件属性是文件类型、文件长度、文件主、文件的许可权(例如,其他用户能否访问该文件)及文件最后的修改时间等。当创建一个新目录时,系统自动创建了两个文件名:.(称为点)和..(称为点-点)。点表示当前目录,点-点表示父目录。在最高层次的根目录中,点-点与点相同。某些 UNIX 文件系统限制文件名的最大长度为 14 个字符,BSD 版本则将这种限制扩展为 255 个字符。

UNIX 文件系统具有如图 3-37 所示的结构。引导块:位于文件系统的开头,占一个物理块,其中包含引导代码段;超级块:描述文件系统的状态,如容量、空闲块号和空闲索引节点号(i_node)等;索引节点区:第一个索引节点是文件系统的根索引节点。当执行了 mount 命令之后,该文件系统的目录结构就可以从这个根索引节点开始进行存取了;数据存储区:存放数据的区域。

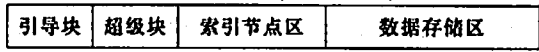


图 3-37 文件系统的布局

进程可以通过系统调用访问文件。例如,Open(打开文件)、Close(关闭文件)、Write(写文件)、Read(读文件)、Stat(查询文件属性)、Chmod(改变文件的许可权)、Chown(改变文件所有者)、Creat(创建一个文件)、Mkdir(创建一个目录文件)、Cd(改变当前目录)、Link(建立连接)和 Unlink(删除文件连接)等。

3. 进程与存储管理

1) 进程的组成

在 UNIX 中,进程由控制块 PCB、正文段和数据段组成。其中进程控制块 PCB 包括常驻主存的基本进程控制块 proc 和非常驻主存的进程扩充控制块 user 结构。正文段可供多个进程执行。为了管理可共享的正文段,UNIX 设置了一张正文表 text[],每个正文段都占据一个表项,用来指明正文段在主存和磁盘的位置。数据段是进程执行时用到的数据段。

2) 进程控制

UNIX 中的进程控制子系统负责进程同步、进程间通信、存储管理及进程调度。控制进程的系统调用有 Fork(创建一个子进程)、exec(改变执行程序的映像)、exit(结束一个进程的执行)、wait(暂停进程的执行,用于进程之间的同步,如父进程等待子进程执行结束)、signal(控制进程对特别事件的响应)、kill(发送软中断信号)、msgsnd(发送消息)和 msgrcv(接受消息)等。

3) 进程调度

UNIX 系统对进程的调度采用动态优先数调度算法,进程的优先数随进程的执行情况而变化。就绪进程是否能占用处理机的优先权取决于进程的优先数,优先数越小优先权越大。UNIX 系统中优先数的确定方法有两种:设置方法和计算方法。设置方法用于要进入睡眠的进程。当进程正在或即将转入用户态运行时,用计算方法确定优先数。UNIX 计算优先数的公式为:

$$p\text{-pri} = p\text{-cpu}/2 + PUSER + p\text{-nice} + NZERO$$

其中, $p\text{-pri}$ 表示进程的优先数, $p\text{-cpu}$ 为处理器的占用时间, $PUSER$ 和 $NZERO$ 表示偏置常数, $p\text{-nice}$ 允许用户使用 shell 命令 nice 或系统调用 nice 修改。用户有权将其值设置为 0 到 39 之间的数。

4) 存储管理

UNIX 早期的版本采用“对换技术”扩充主存容量,进程可以被换出到对换区,也可以从对换区换进主存。高版本的 UNIX 主存管理采用分页式虚拟存储机制,对换技术作为一种辅助手段。采用二次机会页面替换算法。

4. 设备管理

在 UNIX 系统中,文件等于系统中可用的任何资源。UNIX 的设计者们遵循这样一

条规则：UNIX 系统中可以使用的任何计算机资源都用一种统一的方法表示。他们选择用“文件”这个概念作为一切资源的抽象表示方法。

在 UNIX 文件系统中，尽管我们可以看到很多文件，但这些文件并不一定是磁盘上的某些数据集合。实际上，UNIX 系统中的每一个设备，如鼠标、键盘、显示器、磁盘上的分区和打印机等，都在 UNIX 的文件系统中占用一个索引节点。这个节点不仅有名字，而且有创建日期和访问权限等。总而言之，从表面上看，这些节点与一般的磁盘文件没有任何区别，但它们是特殊文件（设备文件）。在编程时，可以通过一个通用的文件描述符（FileDescriptor）对它们进行访问。这种“通过文件描述符访问资源”的观念被扩展到 BSD Socket 当中，成为网络资源访问的标准方式。

UNIX 系统包括两类设备：块设备和字符设备。用户可以通过文件系统与设备接口，因为每个设备都有一个文件名，可以像文件那样存取。设备文件有一个索引节点，在文件系统目录中占据一个节点，但其索引节点上的文件类型与其他文件不同——是“块”或者是“字符”特殊文件。文件系统与设备驱动程序的接口是设备开关表。硬件与驱动程序之间的接口是控制寄存器与 I/O 指令。一旦出现设备中断，根据中断矢量转到相应的中断处理程序，完成用户所要求的 I/O 任务。UNIX 设备管理的主要特点如下。

- 块设备与字符设备具有相似的层次结构。这是指对它们的控制方法以及采用的数据结构和层次结构几乎相同。
- 将设备作为一个特殊文件，并赋予一个文件名。这样，对设备的使用类似于对文件的存取，具有统一的接口。
- 采用完善的缓冲区管理技术。引入“预先读”、“异步写”和“延迟写”方式，进一步提高系统效率。

在 UNIX 系统中，每类设备都有一个驱动程序，用它来控制该类设备。任何一个驱动程序通常都包含用于执行不同操作的多个函数，如打开、关闭和启动设备，以及读和写等函数。为使核心能方便地转向各函数，系统为每类设备提供了一个设备开关表。开关表是每个设备驱动程序的一系列接口过程的入口表，给出了一组标准操作的驱动程序入口地址，文件系统可通过开关表中各函数的入口地址转向适当的驱动程序入口。如图 3-38 所示。图中显示了 UNIX 设备驱动程序通过相应的块设备开关表和字符设备开关表描述与文件系统的接口。

每类设备都有自己的设备处理程序。一般可分成两部分：用于启动设备的设备驱动程序和负责处理 I/O 操作的设备中断处理程序。

5. 输入输出转向

Shell 既是一种命令语言，又是一种程序设计语言。在 UNIX 中，任何一个存放一条或多条命令的文件称为 Shell 程序或 Shell 过程。

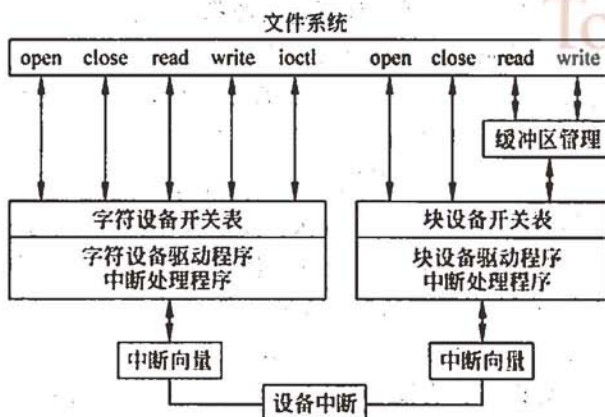


图 3-38 UNIX 系统中的设备开关表

1) 输入输出转向

UNIX 系统的 Shell 向用户提供了输入输出转向命令,可以在不改变应用程序本身的情况下自由地改变其数据的输入源和输出目的地。其中,“>”、“>>”表示输出转向,“<”表示输入转向。例如,cat 命令用来将输入文件的数据显示在屏幕上:

```
cat input.txt
```

上述 cat 命令将 input.txt 文件中的内容输出到屏幕(标准输出设备)上。但是,如果我们将命令写成:

```
cat input.txt > output.txt
```

那么 cat 命令就会将原本输出到屏幕上的内容输入到文件 output.txt 中,并覆盖 output.txt 的内容。如果使用

```
cat input.txt >> output.txt
```

则将 input.txt 文件的内容添加到 output.txt 文件的末尾。

如果使用 cat 命令时没有指定输入文件,cat 就要求用户在键盘(标准输入设备)上输入数据,直到用户输入 Ctrl-D 为止。但是,如果我们使用输入重定向符,就可以写成:

```
cat < input.txt
```

这样,cat 就不再从键盘上而是从 input.txt 文件中读取内容了。实际上,这条命令与 cat input.txt 在效果上是一样的,但它们代表的意义不同。没有使用重定向符的 cat 命令知道自己是在从文件中读取数据;而使用了重定向符的 cat 命令实际上并不知道自己得到

的数据来自 input.txt,它只知道自己是从标准输入设备上读入信息的。Shell 通过重定向耍了个花招,“骗”过了 cat 命令,使它接收到的数据被调了包。输出重定向也一样,cat 命令在输出时也仅知道自己是向标准输出设备输出,而 Shell 用 output.txt 文件掉换了 cat 命令的标准输出设备,使得 cat 命令的输出转到了文件当中。

2) 管道

在 UNIX 中,“|”表示管道。一个管道总是连接两个命令,将左边命令的标准输出与右边命令的标准输入相连。于是,左边命令的输出结果直接成了右边命令的输入。这个功能使用户可以在不改动程序本身的前提下使多个程序可以通过标准输入输出设备进行数据传递。利用管道命令可以简化命令行的写法,例如,下面的三条命令:

```
ls > file
sort < file1 > file2
pr < file2
```

可以用管道命令表示为:

```
ls | sort | pr
```

例如,如果我们要统计当前目录中所有文件和目录的数目,并将其记录在文件 output.txt 中,管道命令如下:

```
ls -a | wc -l > output.txt
```

ls -a 用来列出当前目录下的所有文件和目录;wc -l 负责统计 ls 输出中的行数;最后将输出重定向到 output.txt 中。命令十分简单,完成的工作却相当复杂。

6. Shell 程序

Shell 不但负责管理命令行界面,而且 Shell 自己也是一个编程环境。实际上,我们可以将命令按照命令行的格式写入一个文件,再将其权限设置为可执行,就可以像普通命令一样执行它了。这个文件我们通常称为脚本(script)。熟悉 DOS 的用户也许会脱口而出:这不是批处理文件吗?没错,Shell 脚本就相当于 DOS 的批处理文件。而且 Shell 脚本中也同样支持如 if、for 和 case 等程序控制流程,甚至还支持变量和函数定义!所以才有这样的说法——Shell 实际上是一种编程语言。利用 Shell 语言可以编写出功能很强的 Shell 程序,将程序段组合起来。

1) 正则表达式

在 UNIX 中,正则表达式不仅用在 vi 中,还用在 Shell 中。正则表达式用来确定字符串模式的一个规则集,是对文本字符串的一种描述,该描述能简洁而又完整地刻画文本字符串的关键特性。因此,正则表达式通常被用作字符串的匹配操作。正则表达式符号如

表 3-4 所示。

表 3-4 正则表达式符号

符号	含 义
.	能行内与除换行符之外的任何字符匹配
*	匹配前一字符的零次或多次出现
[]	[]中只能匹配一个字符,若要匹配多个,则使用多个[]
^	如果方括号中的第一个字符是^,则匹配不属于[]中的字符
\$	如果出现在正则表达式末尾,则表示行尾,\$前面的正则表达式所匹配的字符串仅出现在行尾才匹配
\	转义符,用于改变特殊符号的含义,也可后跟一字符的八进制表示
" "	双引号内的字符在匹配时忽视其特殊含义
\<	字首匹配
\>	字尾匹配

【例 3.11】 给出匹配字符串 32787、188567 和 1234567890 的正则表达式。

解: 这些字符串的共同特性都是数字,若要匹配任意长度的数字序列可用如下正则表达式:

[0-9][0-9]* (而不能使用[0-9]*)

【例 3.12】 分别写出字符串 what 出现在行首和行尾的正则表达式。

解: /[^]what 当 what 出现在行首时才会被找到

/ what\$ 当 what 出现在行尾时才会被找到

若匹配文件的任意一行应用为[^]. * \$。

2) shell 变量

shell 变量可分为三种类型: 用户定义变量、系统定义变量和 shell 定义变量。

用户定义变量必须以字母或下划线开始,可以包含字母、下划线和数字的字符序列。用户定义的 shell 变量能用赋值语句置初值或重置值。例如: ux=UNIX。

系统定义变量如表 3-5 所示。

表 3-5 常用的系统定义变量

变量名	含 义
HOME	用户主目录名
PATH	定义 shell 在寻找命令时使用的查找路径
PS1	系统基本提示符,缺省为\$
PS2	系统辅助提示符,缺省为>

变量名	含 义
IFS	内部字段分割符,缺省是空格、制表符和换行符
MAIL	存放用户邮件文件的路径名
TERM	定义用户使用的终端类型
CDPATH	cd 命令要查找的目录表
LOGNAME	用户的注册名
SHELL	Shell 程序的路径名
MANPATH	连接动态库时的搜索路径

shell 定义变量如表 3-6 所示。

表 3-6 shell 定义变量

变量名	含 义
\$0	命令名,在 shell 程序内可用 \$0 获得该程序的名字
\$1~\$9	shell 程序的位置参量
\$#	位置参数的个数,不包括命令名
\$*	所有位置参量,即相当于 \$1,\$2,...
@	与 \$* 基本相同,但当用双引号转义时,“@”能分解成多个参数,但“\$*”则合并成一个参数
?	上一命令的返回代码。成功返回 0,否则返回 1。
\$	当前命令的进程标识数
!	最近执行的后台进程标识数
-	Shell 标识位组成的字符串,可由 Shell 传递,或由 set 命令设置

3) Shell 程序

Shell 向用户提供了许多用于简化输入的符号,这些符号包括各种通配符、字符串定义符、转义符及变量定义符等。这些符号可以看作 Shell 的保留字,通常称为“元字符”。元字符的种类和作用非常多,它们无论在 Shell 的命令行输入还是在 Shell 程序设计中都起着非常重要的作用。

【例 3.13】 显示用户登录名、用户主目录以及当前命令进程标识符的 Shell 程序如下:

```
echo username; $ LOGNAME
echo Home directory; $ HOME
echo Current shell's PID; $ $
```

有趣的是,Shell 命令行本身也是一个交互式的脚本执行环境。也就是说,在命令行上同样可以使用脚本中的控制语句,也可以定义变量(实际上就是环境变量),甚至可以定义函数,而且与脚本文件中的命令一样。但是有一点必须注意:Shell 程序有许多种,不同的 Shell 有不同的编程命令和语法。

3.8.2 Windows 2000/XP 操作系统

1. Windows 操作系统的体系结构

和许多操作系统一样,Windows 操作系统也通过硬件机制实现核心态(管态)与用户态(目态)两种状态。当处于核心态时,可以执行任何指令,并改变状态。当处于用户态时,只能执行非特权指令。用户程序一般都运行在用户态,而操作系统中至关重要的代码则运行在核心态,以免被错误的应用程序破坏。

Windows 2000/XP 操作系统的核心组件使用了面向对象的设计原则。例如,它不能直接访问某个数据结构中由单个组件维护的消息,这些组件只能使用外部接口以参数传递方式访问或修改这些数据。Windows 2000/XP 操作系统的体系结构框架如图 3-39 所示。

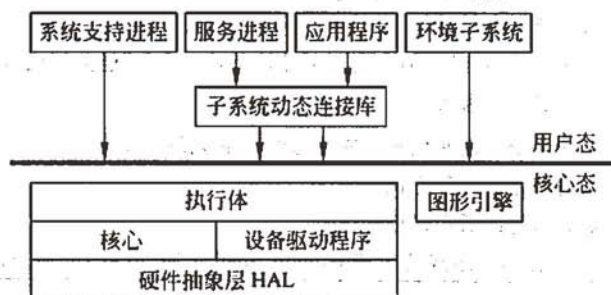


图 3-39 Windows 2000/XP 操作系统体系结构

用户进程有 4 种类型:

(1) 系统支持进程,如登录进程 WINLOGO 和会话管理器 SMSS。它们不是 Windows 2000/XP 的服务;

(2) 服务进程,是 Windows 2000/XP 的服务,如事件日志服务;

(3) 环境子系统,它们向应用程序提供运行环境(操作系统功能调用接口)。Windows 2000/XP 有 3 个环境子系统: Win32、POSIX 或 OS/21.2;

(4) 应用程序,它们可以是 Win32、Windows 3.2、MS-DOS、POSIX 或 OS/21.2 5 种类型。

从图中可以看出,服务进程和应用程序是不能直接调用操作系统服务的,必须通过子

系统动态连接库与系统交互。

Windows 2000/XP 核心组件包括：

- 核心,包含最低层的操作系统,例如,线程调度、中断和异常和多处理器同步等,同时它也提供执行体,用于实现高级结构的一组例程和基本对象;
- 执行体,包含基本的操作系统服务,例如,主存管理、进程和线程管理、安全控制、I/O 以及进程间的通信;
- 硬件抽象层,将内核、设备驱动程序以及执行体与硬件分隔开来,以便适应多种平台;
- 设备驱动程序,包括文件系统和硬件设备驱动程序。其中,硬件设备驱动程序将用户的 I/O 函数调用转换为对特定硬件设备的 I/O 请求;
- 图形引擎,包含实现图形用户界面的基本函数。

2. 文件系统

Windows 2000/XP 支持传统的 FAT 文件系统,该文件系统最初是针对小容量的磁盘而设计的。但随着计算机外存容量的迅速扩展,出现了明显的不适应性。FAT 文件系统最多只能容纳 2^{12} 或 6^{16} 个簇,单个文件卷的容量小于 2GB。如果一个簇包含的扇区数增加,可以使单个卷的容量增大,但是文件空间的碎片很多,浪费很大。

从 Windows 9X 和 Windows ME 开始,FAT 表被扩展到了 32 位,形成了 FAT32 文件系统,解决了 FAT16 系统在文件容量上的问题,可以支持 4GB 的大硬盘分区。但由于 FAT 表的大幅度扩充,造成文件系统效率大幅度下降。Windows 98 支持 FAT32 文件系统,但与其同期开发的 Windows NT 不支持 FAT32。基于 NT 构建的 Windows 2000/XP 支持 FAT32 系统。除此之外,Windows 2000/XP 支持的文件系统包括 CDFS、UDF、FAT12、FAT16、FAT32 和 NTFS。

NTFS 文件系统是 Windows 2000/XP 本身的文件系统格式。NTFS 使用 64 位簇进行索引,这使得 NTFS 有能力寻址达 16 exabytes(160 亿 GB)。NTFS 的主要特征如下:

- 可恢复性:之所以建立新的 Windows 文件系统,就是为了具备从系统崩溃和磁盘故障中恢复数据的能力。当发生故障时,NTFS 能够重建文件卷,并使他们恢复到一个一致的状态。
- 安全性:NTFS 使用对象模型来实施安全机制。一个打开的文件是作为一个文件对象实现的,该文件对象有一个作为该文件一部分而存储在磁盘上的安全描述体。在进程试图打开一个文件对象的句柄之前,NTFS 安全系统会验证该进程是否具有资格。安全描述体与用户登录到系统的身份的結合保证了除非系统管理员或文件的所有者给定了特定的许可,否则进程不能访问该文件。
- 大磁盘和大文件:NTFS 比 FAT 以及其他大多数文件系统都能够更有效地支持

非常大的磁盘和非常大的文件。

- **多数据流：**在 NTFS 中，每一个与文件有关的信息单元，如文件名、所有者、时间标记和数据等都可以作为文件属性来执行，所以 NTFS 文件可以包含多数据流。这项技术为高端服务器应用程序提供了功能强大的新手段。
- **通用索引功能：**NTFS 的体系结构允许在一个磁盘卷中索引文件属性，从而可以有效地定位和匹配各种标准文件。在 Windows 2000/XP 中，这种索引机制被扩展到其他属性，如对象 ID。

Windows 2000/XP 还提供分布式文件服务。分布式文件系统 (DFS) 是用于 Windows 2000/XP 服务器上的网络组件。最初它是作为一个扩展层出现在 Windows NT 上的，但在功能上受到限制。在 Windows 2000/XP 中，这种限制得到了修正。DFS 使用户更容易找到和管理网上的数据。使用 DFS，可以更加容易地创建单目录树，该目录树可以包括多文件服务器和组、部门或企业中的文件共享。

3. 进程

在 Windows 2000/XP 中，进程是资源分配的单位，并将进程作为对象管理。可以通过相应的句柄引用对象。操作系统提供一组控制进程对象的服务。进程对象属性包括进程标识、资源访问令牌和进程的基本优先级等。

Windows 2000/XP 的线程是内核线程，是处理机调度的单位。线程的上下文主要包括寄存器、线程环境块、核心栈和用户栈。Windows 2000/XP 中的线程有 7 种状态，如图 3-40 所示。

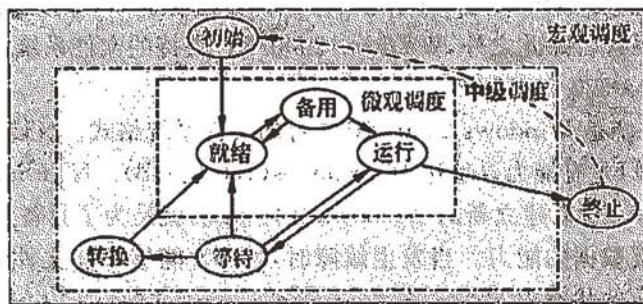


图 3-40 Windows 2000/XP 中线程的状态

就绪状态：已获得除处理机之外的所需资源，正等待调度。**备用状态：**已选择好线程的执行处理机，正等待描述表切换，以进入运行态。系统中每个处理机只能有一个处于备用状态的线程。**运行状态：**正在处理机上运行的线程，直到被抢先、时间片用完、线程终止或进入等待。**等待状态：**线程正等待某对象，以同步线程的执行。当等待时间出现

时,根据优先级进入运行或就绪状态。转换状态:该状态与就绪状态类似,但线程的内核堆栈位于外存。终止状态:线程执行完毕就进入终止状态。执行体有一个指向线程对象的指针,可以将处于终止状态的线程重新初始化,并再次使用。初始化状态:线程创建时的状态。

4. 存储管理

Windows 2000/XP 默认情况下使用二级页面表结构来转换物理地址和虚拟地址。32 位的虚拟地址被解释为三个单独的组件:页面目录索引,页面表索引和字节索引,并被当作描述页面映射的结构的索引,如图 3-41 所示。页大小和 PTE 宽度表明页面目录和页面表索引字段的宽度。例如,在 X86 系统上,因为页面为 4096 字节,字节索引则为 12 位。

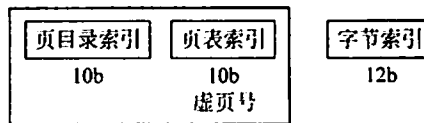


图 3-41 X86 系统中一个 32 位虚拟地址结构

“页面目录索引”用来指明虚拟地址的页目录在页表中的位置,“页表索引”用来确定页表项 PTE 在页表中的具体位置,PTE 包括了虚拟地址被映射到的物理地址。“字节索引”用于在该物理页面内寻找正确的地址。图 3-42 显示了这 3 个值之间的关系和如何把虚拟地址映射到物理地址的过程。

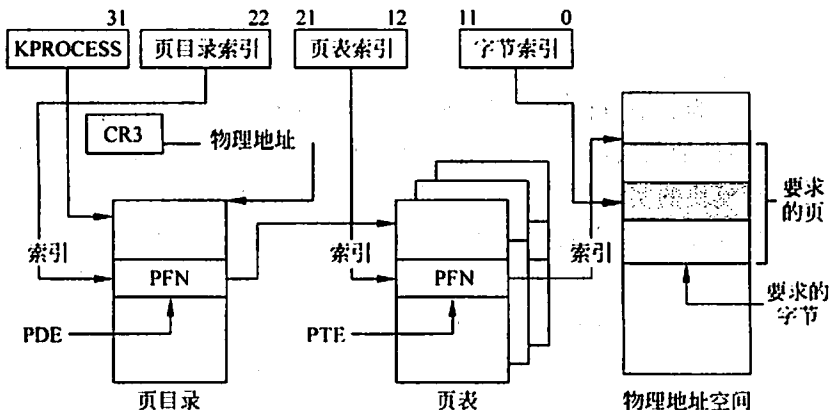


图 3-42 虚拟地址变换

以下是转换一个虚拟地址的基本步骤。

① 主存管理硬件定位当前进程的页目录。每次进程切换时,一般是通过操作系统设置专用的 CPU 寄存器来通知硬件设备新进程页目录的地址。

② 页目录索引用来在页目录中查找页目录项(Page Directory Entry,PDE)的索引,页目录项描述了映射虚拟地址时需要的页面表位置。PDE 包括页面表的页框号(Page Frame Number,PFN)。

③ 页表索引用来在页表中指出 PTE 的位置。PTE 描述了虚拟页面在物理主存的位置。

④ PTE 用来确定页框的位置。如果所需的页面有效,则页表项包含物理主存中的一个页框号 PFN。如果页面无效,则主存管理错误处理程序将查找页面并试图使页面有效。如果不能使页面有效,错误处理程序将产生一个访问违例或错误检查。

⑤ 当 PTE 指向一个有效的页面时,字节索引用来查找在物理页面内所需数据的地址。

5. 设备管理

Windows 2000/XP 操作系统采用和强调了软件工程中抽象的原则,在设计中全力找出各种事务的共性,用一致的模型、方法和界面来实现规范化,如采用客户机/服务器模型规范各个用户进程之间的关系。尤其突出的是建立了广义的资源管理概念,并统一地用对象模型来描述和规范化,使系统的复杂性降低。在输入输出设计上,建立了一个统一一致的高层界面。IO 设备虚拟界面。即将所有的读写数据看成直接送往虚拟文件的字节流。

Windows 2000/XP 的 I/O 系统体系结构如图 3-43 所示,它由几个可执行模块和大量设备驱动程序组成。

从图 3-43 中可以看到,Windows 2000 的 I/O 体系设计使用了分层结构,这有利于实现其平台无关性,也为其他目标的实现带来了便利。在整个体系的最底层也就是直接与硬件交互的层是硬件抽象层(HAL),它隐藏了不同硬件平台之间的差异,使得上层的驱动程序、各可执行模块的实现与平台无关。从具体实现的角度讲,HAL 实际上是由系统提供的许多总线设备驱动程序的集合,尽管具体的总线可能不同,但它们向上层提供了统一的接口。

在 HAL 层之上的是设备驱动层,在这个层次上,各驱动程序利用总线驱动程序提供的 I/O 端口访问函数访问并控制对应的硬件设备。事实上,设备驱动层又可分为许多更细的层次。

在设备驱动层之上是 I/O 系统层,它由一系列管理器组成,如 I/O 管理器、电源管理器、即插即用(PnP)管理器和 WMI(Windows Management Instrument)等。它们负责用户模式的程序与核心模式的驱动程序的交互控制、即插即用及资源分配等一系列工作。

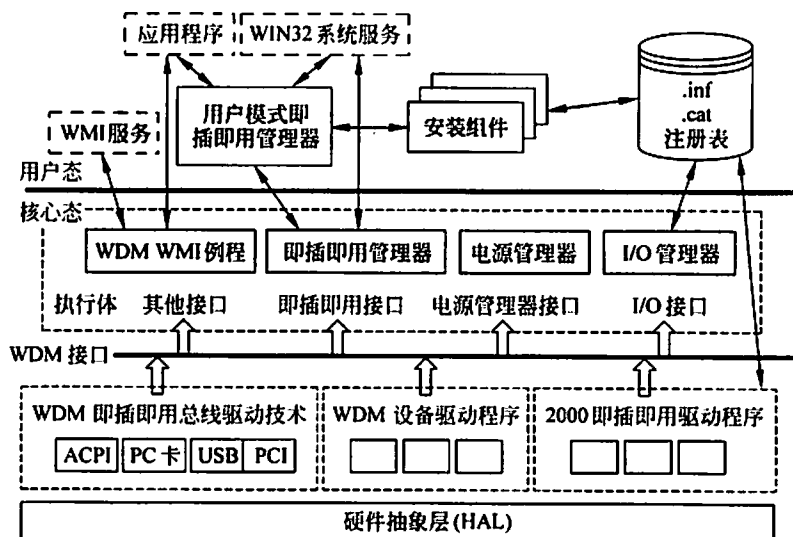


图 3-43 Windows 2000/XP 的 I/O 系统体系结构

图 3-43 中使用双箭头表示 I/O 系统层与设备驱动层的关系。但实际上 I/O 系统层也可直接与 HAL 通信,例如,PnP 管理器就直接与总线驱动合作,来检测并响应设备的加入和移除。

以上这些模块都运行在 Windows 2000/XP 的核心态,它们是操作系统执行体的一部分,操作系统对他们的错误操作没有保护机制,也就是说,如果这些模块出现问题,整个系统就会崩溃。在这 3 层中,开发者一般只能编写驱动程序模块并把它加入核心。

第 4 章 程序设计语言基础

程序设计语言是为了书写计算机程序而人为设计的符号语言,用于对计算过程进行描述、组织和推导。程序设计语言的广泛使用始于 1957 年,经过四十多年的发展,目前世界上流行的程序设计语言有上百种之多,程序设计语言的演化速度已经超越了运行它们的机器。

4.1 基础知识

本节主要介绍程序设计语言的基本概念、基本成分和一些有代表性的程序设计语言的特点及其适用范围。

4.1.1 程序设计语言的基本概念

1. 低级语言和高级语言

计算机问世之初,计算机的硬件只能识别由 0、1 字符串组成的机器指令序列,即机器指令程序,因此机器指令程序是最基本的计算机语言。由于机器指令是特定的计算机系统所固有的面向机器的语言,所以用机器语言进行程序设计时,需要对机器结构有较多的了解。另外,用机器语言编制出来的程序可读性很差,也难以理解、修改和维护。为了提高程序设计的效率,人们就用容易记忆的符号代替 0、1 序列表示机器指令中的操作码和操作数。例如,用 ADD 表示加法, SUB 表示减法等。用符号表示的指令称为汇编指令,汇编指令的集合称为汇编语言。虽然使用汇编语言编写程序的效率和程序的可读性有所提高,但汇编语言是一种和计算机的机器语言十分接近的语言,它的书写格式在很大程度上取决于特定计算机的机器指令。因此它仍然是一种面向机器的语言,通常称机器语言和汇编语言为低级语言。在这个基础上,人们开发出功能更强、抽象程度更高的语言以提高效率,于是就产生了面向各类应用的程序语言。目前已开发出许多高级语言,常见的有 FORTRAN、COBOL、Pascal、C、Ada、C++ 和 Java 等。这类语言与人们使用的自然语言比较接近,大大提高了程序设计的效率。

2. 编译程序和解释程序

目前,尽管人们可以借助高级语言与计算机进行交互,但是计算机仍然只能理解和执行由 0、1 序列构成的机器语言,因此高级程序语言需要翻译,担负这一任务的程序称为语

言处理程序。由于应用的不同,语言之间的翻译也是多种多样的。它们大致可分为:汇编程序、解释程序和编译程序。

用某种高级语言或汇编语言编写的程序称为源程序,源程序不能直接在计算机上执行。如果源程序是用汇编语言编写的,则需要一个称为汇编程序的翻译程序将其翻译成目标程序后才能执行。如果源程序是用某种高级语言编写的,则需要相应的解释程序或编译程序对其进行翻译,然后在机器上运行。

解释程序也称为解释器,它或者直接解释执行源程序,或者将源程序翻译成某种中间表示形式后再执行。而编译程序(编译器)则是将源程序翻译成目标语言程序,然后在计算机上运行目标程序。两种语言处理程序的根本区别是:在编译方式下,机器上运行的是与源程序等价的目标程序,源程序和编译程序都不再参与目标程序的执行过程;而在解释方式下,解释程序和源程序(或其某种等价表示)要参与到程序的运行过程中,运行程序的控制权在解释程序。解释器翻译源程序时不生成独立的目标程序,而编译器则需将源程序翻译成独立的目标程序。

3. 程序设计语言的定义

一般地,程序设计语言的定义都涉及语法、语义和语用等3个方面。

语法是指由程序语言基本符号组成程序中的各个语法成分(包括程序)的一组规则,其中由基本符号构成的符号(单词)书写规则称为词法规则,由符号(单词)构成语法成分的规则称为语法规则。程序语言的语法可通过形式语言进行描述。

语义是程序语言中按语法规则构成的各个语法成分的含义,可分为静态语义和动态语义。静态语义指编译时可以确定的语法成分的含义,而运行时刻才能确定的含义是动态语义。一个程序的执行效果说明了该程序的语义,它取决于构成程序的各个组成部分的语义。

语用表示了构成语言的各个记号和使用者的关系,涉及符号的来源、使用和影响。

语言的实现则有个语境问题。语境是指理解和实现程序设计语言的环境,这种环境包括编译环境和运行环境。

4.1.2 程序设计语言的种类与特点

1. 程序语言的发展概述

程序语言有交流算法和计算机实现的双重目的。现在的程序语言种类繁多,它们在实际应用上各有不同的侧重面。

若一种程序语言不依赖于机器硬件,则称为高级语言。若程序语言能够应用于范围广泛的问题求解过程中,则称之为通用的程序设计语言。

FORTRAN 是第一个被广泛用来进行科学计算的高级语言。一个 FORTRAN 程序

由一个主程序或一个主程序与若干个子程序组成。主程序及每一个子程序都是独立的程序单位,称为一个程序模块。在 FORTRAN 中,子程序是实现模块化的有效途径。

ALGOL 60 主导了 20 世纪 60 年代程序语言的发展。它有严格的语法规则,采用巴克斯范式 BNF 来描述语言的语法。ALGOL 是一个分程序结构的语言。一般来说,一个 ALGOL 程序本身就是一个分程序。每个分程序由 begin 和 end 括起来,以说明分程序的范围和它所管辖的名字的作用域。分程序的结构可以是嵌套的,也就是说,分程序内可以包含别的分程序。过程也可以称为一个分程序。同一个名字在不同的分程序中可以代表完全不同的实体。如果一个名字在若干层嵌套分程序中多次被说明,则程序中该名字的使用由离使用点最近的内层说明决定,即“最近嵌套原则”。此外,ALGOL 还提供了数组的动态说明和过程的递归调用。

因为一个分程序只有在执行时才需要数据空间,执行完成后就释放所占用的空间。因此,分程序结构的主要优点是可以非常有效地使用存储器。由于分程序结构的嵌套性,用一个栈来组织和管理整个程序运行时的数据空间是非常方便的。

COBOL(COMmon Business Oriented Language)是一种面向事务处理的高级语言。在企业管理中,数值计算并不复杂,但数据处理的信息量却很大。1959 年,由美国的一些计算机用户组织设计了专用于商务处理的计算机语言 COBOL,并于 1961 年由美国数据系统语言协会公布。经不断修改、丰富、完善和标准化,已发展了多种版本。

COBOL 语言使用了 300 多个英语保留字,大量采用普通英语词汇和句型。COBOL 语言的语法规则很严格。目前 COBOL 语言主要应用于情报检索及商业数据处理等领域。

Pascal 语言是一种结构化程序设计语言,由瑞士苏黎世联邦工业大学的沃斯(N. Wirth)教授研制,1971 年正式发表。Pascal 是从 ALGOL60 衍生的,但功能更强且容易使用。Pascal 语言在高校计算机软件教学中曾一直处于主导地位。后来的 Pascal 语言中添加了并发控制结构,产生了并发 Pascal。在 Pascal 语言中分程序和过程两个概念合二为一,统一为过程。而一个 Pascal 程序本身可看作一个操作系统调用的过程。Pascal 过程是可以嵌套和递归的。

C 语言是 20 世纪 70 年代发展起来的一种通用程序设计语言,它提供了一个丰富的运算符集合以及比较紧凑的语句格式。C 语言的主要特色是兼顾了高级语言和汇编语言的特点,简洁,丰富,可移植。C 与 UNIX 操作系统紧密相关,UNIX 操作系统及其上的许多软件都是用 C 编写的。C 提供了高效的执行语句,并且允许程序员直接访问操作系统和底层硬件,这使得 C 在系统应用和实时处理中成为主要语言。

C++ 是在 C 语言的基础上于 20 世纪 80 年代发展起来的,与 C 兼容。在 C++ 中,最主要的是增加了类机制,使其成为一种面向对象的程序设计语言。

Java 产生于 20 世纪 90 年代,其目的是用于开发网络浏览器的小应用程序,但作为一种通用的程序设计语言,Java 也得到了广泛的应用。Java 保留了 C++ 的基本语法、类和继承等概念,删掉了 C++ 中一些不好的特征,因此与 C++ 相比,Java 更简单,其语法和语义更合理。

各种程序语言都在不断地发展之中。目前,程序设计语言及编程环境正向面向对象及可视化编程环境方向发展,出现了许多新的语言及其开发工具。例如,微软公司的 Visual 系列编程工具及 Power Builder 等,已经得到了广泛的应用。

2. 程序语言种类和特点

程序语言的分类没有统一的标准,这里根据程序设计的方法将程序语言大致分为命令式程序设计语言、面向对象的程序设计语言、函数式程序设计语言和逻辑型程序设计语言等类型。

1) 命令式程序设计语言

命令式语言是基于动作的语言,在这种语言中,计算被看成动作的序列。命令式语言族开始于 FORTRAN, Pascal 和 C 语言体现了命令式程序设计的关键思想。

2) 面向对象的程序设计语言

面向对象的程序设计在很大程度上应归功于从模拟领域发展起来的 Simula, Simula 提出了对象和类的概念。C++、Java 和 smalltalk 是面向对象程序设计语言的代表。

一般认为,面向对象程序语言主要包含下面几个概念。

(1) 对象:对象可以是人们要进行研究的任何事物,它具有状态和操作。面向对象语言把状态和操作封装于对象实体之中,并提供一种访问机制,使对象的“私有数据”仅能由这个对象的操作来访问。用户只能通过向允许公开的操作提出要求(或发送消息),才能查询和修改对象的状态。这样,对象状态的具体表示和操作的具体实现都被隐藏起来了。

(2) 类:类是面向对象语言必须提供的、由用户定义的数据类型,它将具有相同状态、操作和访问机制的多个对象抽象成一个对象类。在定义了类以后,属于这种类的一个对象叫作类实例或类对象。类代表一般,而类的一个对象代表具体。

(3) 继承:继承是面向对象语言的另一个基本要素。在客观世界中,存在着整体和部分、一般和特殊的关系。继承实现了一般与特殊的关系,解决了软件的重用性和扩充性问题。类与类之间可以组成继承层次,一个类的定义可以定义在另一个已定义类的基础上,前者称为子类,后者称为父类。子类可以继承父类中的属性和操作,也可以定义自己的属性和操作,从而使内部表示上有差异的对象可以共享与它们结构中的共同部分有关的操作,达到概念复用和代码重用的目的。

3) 函数式程序设计语言

函数式语言是一类以 λ -演算为基础的语言,其基本概念来自于 LISP 语言,这是一个在 1958 年为了人工智能应用而设计的语言。函数是一种对应规则(映射),它使定义域中的每个元素和值域中惟一的元素相对应。例如:

函数定义 1: $\text{Square}[x] := x * x$

函数定义 2: $\text{Plustwo}[x] := \text{Plusone}[\text{Plusone}[x]]$

函数定义 3: $\text{fact}[n] := \text{if } n=0 \text{ then } 1 \text{ else } n * \text{fact}[n-1]$

函数定义 2 中使用了函数复合,即将一个函数调用嵌套在另一个函数定义中。在函数定义 3 中,函数被递归定义。由此可见,函数可以看成一种程序,其输入就是定义在左边括号中的变量。可以将输入组合起来产生一个规则,组合过程中也可以使用其他函数或该函数本身。这种用函数和表达式建立程序的方法就是函数式程序设计。函数型程序设计语言的优点之一就是表达式中出现的任何函数都可以用其他函数来代替,只要这些函数调用产生相同的值。

函数式语言的代表 LISP 在许多方面与其他语言不同,其中最为显著的是,该语言中程序和数据的形式是等价的,这样数据结构就可以作为程序执行,同样程序也可以作为数据修改。在 LISP 中,大量地使用递归。

4) 逻辑型程序设计语言

逻辑型语言是一类以形式逻辑为基础的语言,其代表是建立在关系理论和一阶谓词理论上的 PROLOG。PROLOG 表示“Programming in Logic”。PROLOG 程序是一系列事实、数据对象或事实间的具体关系和规则的集合。通过查询操作把事实和规则输入数据库。用户通过输入查询来执行程序。在 PROLOG 中,关键操作是模式匹配,通过匹配一组变量与一个预先定义的模式并将该组变量赋给该模式来完成操作。以值集合 S 和 T 上的二元关系 R 为例, R 实现后,可以询问:

- 已知 a 和 b ,确定 $R(a,b)$ 是否成立;
- 已知 a ,求所有使 $R(a,y)$ 成立的 y ;
- 已知 b ,求所有使 $R(x,b)$ 成立的 x ;
- 求所有使 $R(x,y)$ 成立的 x 和 y 。

逻辑型程序设计具有与传统的命令型程序设计完全不同的风格。PROLOG 数据库中的事实和规则是一些 Hore 子句。Hore 子句的形式为“ $P: -P_1, P_2, \dots, P_n$ ”,其中 $n \geq 0$, $P_i (1 \leq i \leq n)$ 为形如 $R_i(\dots)$ 的断言, R_i 是关系名。该子句表示规则:若 P_1, P_2, \dots, P_n 均为真(成立),则 P 为真。当 $n=0$ 时, Hore 子句变成“ P ”,这样的子句称为事实。

一旦有了事实与规则后,就可以提出询问。测试用户询问 A 是否成立时,可采用归结方法:

- 如果程序中包含事实“ P ”,且 P 和 A 匹配,则 A 成立。

- 如果程序中包含 Hore 子句“ $P: - P_1, P_2, \dots, P_n$ ”, 且 P 和 A 匹配, 则 PROLOG 转而测试“ P_1, P_2, \dots, P_n ”, 只有当 P_1, P_2, \dots, P_n 都成立时才能断言 P 成立。当求解某个 P_i 失败时, 则返回到前面的某个成功点并尝试另一种选择, 也就是进行回溯。
- 只有当所有可能情况都已穷尽时, 才能推导出 P 失败。

PROLOG 有很强的推理功能, 适用于书写自动定理证明、专家系统以及自然语言理解等问题的程序。

4.1.3 程序设计语言的基本成分

程序设计语言的基本成分包括数据、运算、控制和传输等。

1. 数据成分

程序语言的数据成分指的是一种程序语言的数据类型。数据对象总是对应着应用系统中某些有意义的东西, 数据表示则指定了程序中值的组织形式。数据类型用于代表数据对象, 还用于在基础机器中完成对值的布局, 同时还可用于检查表达式中对运算的应用是否正确。

数据是程序操作的对象, 具有存储类别、类型、名称、作用域和生存期等属性, 使用时要为它分配内存空间。数据名称由用户通过标识符命名, 标识符是由字母、数字和称为下划线的特殊符号“ $_$ ”组成的标记; 类型别说明数据占用内存的大小和存放形式; 存储类别说明数据在内存中的位置和生存期; 作用域则说明可以使用数据的代码范围; 生存期说明数据占用内存的时间范围。从不同角度可将数据进行不同的划分。

1) 常量和变量

按照程序运行时数据的值能否改变, 将数据分为常量和变量。程序中的数据对象可以具有左值和右值。左值指存储单元(或地址、容器), 右值是值(或内容)。变量具有左值和右值, 在程序运行的过程中其右值可以改变; 常量只有右值, 在程序运行的过程中其右值不能改变。

2) 全局量和局部量

按数据的作用域范围, 可分为全局量和局部量。系统为全局变量分配的存储空间在程序运行的过程中一般是不改变的, 而为局部变量分配的存储单元是动态改变的。

3) 数据类型

按照数据组织形式的不同可将数据分为基本类型、用户定义类型、构造类型及其他类型。C(C++) 的数据类型如下所述。

- 基本类型: 整型(int)、字符型(char)、实型(float、double)和布尔类型(bool)。
- 特殊类型: 空类型(void)。
- 用户定义类型: 枚举类型(enum)。

- 构造类型：数组、结构和联合。
- 指针类型：type *。
- 抽象数据类型：类类型。

其中，布尔类型和类类型是 C++ 在 C 语言的基础上扩充的。

2. 运算成分

程序语言的运算成分指明允许使用的运算符及运算规则。大多数高级程序语言的基本运算可以分成算术运算、关系运算和逻辑运算，有些语言如 C(C++) 还提供位运算。运算符的使用与数据类型密切相关。为了确保运算结果的惟一性，运算符要规定优先级和结合性，必要时还要使用圆括号。

3. 控制成分

控制成分指明语言允许表述的控制结构，程序员使用控制成分来构造程序中的控制逻辑。理论上已经证明，可计算问题的程序都可以用顺序、选择和重复这 3 种控制结构来描述。

1) 顺序结构

顺序结构用来表示一个计算操作序列。计算过程从所描述的第一个操作开始，按顺序依次执行后续的操作，直到序列的最后一个操作，如图 4-1 所示。顺序结构内也可以包含其他控制结构。

2) 选择结构

选择结构提供了在两种或多种分支中选择其中一个的逻辑。基本的选择结构是通过指定一个条件 P，然后根据条件的成立与否来决定控制流走程序块 A 还是走程序块 B，从两个分支中选择一个执行，如图 4-2(a) 所示。选择结构中的程序块 A 或程序块 B 还可以包含顺序、选择和重复结构。程序语言中通常还提供简化的选择结构，也就是没有程序块 B 的分支结构（如图 4-2(b) 所示），以及多分支选择结构。

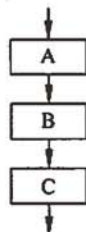


图 4-1 顺序结构示意图

3) 循环结构

循环结构描述了重复计算的过程，通常由 3 部分组成：初始化、需要重复计算的部分和重复的条件。其中初始化部分有时在控制的逻辑结构中并无显式表示。重复结构主要有两种形式：while 型重复结构和 do-while 型重复结构。while 型结构的逻辑含义是先判断条件 P，若成立，则执行需要重复的程序块 A，然后再去判断条件；否则控制流就退出重复结构，如图 4-3(a) 所示。do-while 型结构的逻辑含义是先执行需要重复的程序块 A，然后判断条件 P，若成立则继续执行程序块 A 的过程和条件的判断；否则控制流就退出重复结构，如图 4-3(b) 所示。

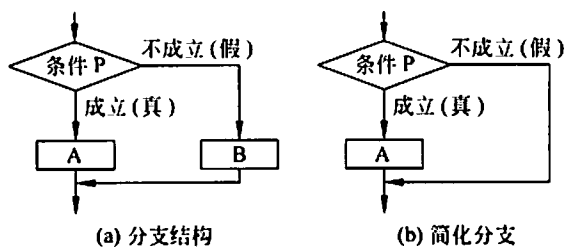


图 4-2 选择结构示意图

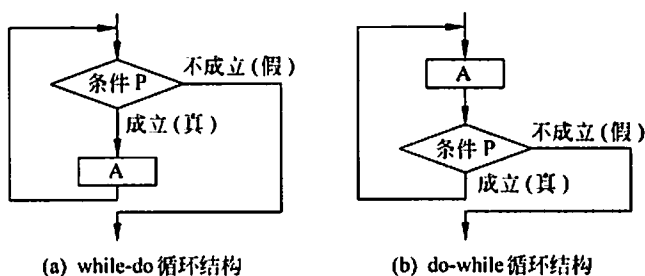


图 4-3 循环结构示意图

4) C(C++) 提供的控制语句

(1) 复合语句。

复合语句用于描述顺序控制结构。复合语句是一系列用“{”和“}”括起来的声明和语句，其主要作用是将多条语句组成一个可执行单元。语法上能出现语句的地方都可以使用复合语句。复合语句是一个整体，要么全部执行，要么一条语句也不执行。

(2) if 语句和 switch 语句。

- if 语句实现的是双分支选择结构，其一般形式为：

if (表达式) 语句 1; else 语句 2;

其中语句 1 和语句 2 可以是任何合法的 C(C++) 语句，当语句 2 为空语句时，可以简化为：if (表达式) 语句 1;

使用 if 语句时，需要注意的是 if 和 else 的匹配关系。C 语言规定，else 总是与离它最近的尚没有 else 的 if 相匹配。

switch 语句描述了多分支的选择结构，其一般形式为：

```
switch (表达式){
    case 常量表达式 1: 语句 1;
```

```
case 常量表达式 2: 语句 2;  
.....  
case 常量表达式 n: 语句 n;  
default: 语句 n+1;  
}
```

执行 switch 语句时,首先计算表达式的值,然后用所得的值与列举的常量表达式值依次比较。若任一常量表达式都不能与所得的值相匹配,则执行 default 的“语句序列 n+1”,然后结束 switch 语句。若此值与常量表达式 $i(i=1,2,\dots,n)$ 的值相同,则执行“语句序列 i”,当“case i”的语句序列 i 中无 break 语句时,则执行随后的语句序列 i+1,语句序列 i+2,……直到语句序列 n+1 执行完成后,才退出 switch 语句,或者以 break 跳出 switch 语句。为使程序在执行完“语句序列 i”后结束整个 switch 语句,在语句序列 i 中应包含控制流能够到达的 break 语句。

表达式可以是任何类型的,常用的是字符型或整型表达式。多个常量表达式可以共用一个语句组。语句组可以包括任何的可执行语句,且无须用“{”和“}”括起来。

(3) 循环语句。

C(C++) 语言提供了 3 种形式的循环语句,用于实现重复型的控制结构。

- while 语句。

while 语句描述了先判断重复条件再执行循环体的控制结构,while 语句的一般形式如下:

```
while (条件表达式) 循环体语句;
```

其中,循环体语句是内嵌的语句,当循环体语句多于一条时,应使用“{”和“}”括起来。执行 while 语句时,先计算条件表达式的值,当其值为非 0 时,就执行循环体语句,然后重新计算条件表达式的值后再进行判断,否则就结束 while 语句的执行过程。

- do-while 语句。

do-while 语句实现了先执行循环体,再判断条件的控制结构,其一般格式如下:

```
do  
    循环体语句;  
while (条件表达式);
```

执行 do-while 语句时,先执行内嵌的循环体语句,然后再计算条件表达式的值。若其值为非 0,则再一次地执行循环体语句,计算条件表达式并作判断。直到条件表达式的值为 0 时,才结束 do-while 语句的执行过程。

- for 语句。

for 语句的基本格式是: for(表达式 1;表达式 2;表达式 3)循环体语句;

for 语句可以用 while 语句等价地表示为

表达式 1:

```
while(表达式 2){
```

```
    循环体语句;
```

```
    表达式 3;
```

```
}
```

for 语句的使用是很灵活的,其内部的 3 个表达式都可以省略,但用于分隔 3 个表达式的分号“;”不能遗漏。

实际上,在 for 语句的语义中,表达式 1 只计算一次,其作用相当于对 for 语句中一些变量进行初始化,因此可以简化 for 语句,将其功能放在 for 语句之前实现即可。表达式 2 的值是判断是否执行循环体语句的依据,如果缺省,则认为该表达式的值为非 0,这意味着 for 语句将无法终止,除非在循环体语句中加入终止循环的 break 语句。表达式 3 用于修改可以改变表达式 2 值的变量,若省略,其功能可放在循环体语句中。

C 语言中还提供了实现控制流跳转的 goto 语句,由于它会破坏程序的逻辑结构,因此不提倡使用。

程序语言的传输成分指明语言允许的数据传输方式,如数据的输入和输出等。

4. 函数

C 程序由一个或多个函数组成,每个函数都有一个名字,其中有且仅有一个名字为 main 的函数,作为程序运行时的起点。函数是程序模块的主要成分,它是一段具有独立功能的程序。函数的使用涉及 3 个概念:函数定义、函数声明和函数调用。

1) 函数定义

函数的定义包括两部分:函数首部和函数体。函数的定义描述了函数做什么和怎么做。函数定义的一般格式如下:

```
返回值的类型  函数名(形式参数表)  //函数首部
```

```
{
```

```
    函数体;
```

```
}
```

函数首部说明了函数返回值的数据类型、函数的名字和函数运行时所需的参数及类型。函数所实现的功能在函数体中进行描述。如果函数没有返回值,则返回值的类型为 void。如果程序中不指定返回值,编译程序默认函数返回一个 int 型的值。函数名是一个标识符,好的编程风格是函数名能反映函数的功能。形式参数表列举了函数要求调用者

提供的参数的个数、类型和顺序,是函数实现功能时所必需的。若形式参数表为空,可用 void 说明。

C(C++)程序中所有函数的定义都是独立的。在一个函数的定义中不允许定义另外一个函数,也就是不允许函数的嵌套定义。

2) 函数声明

函数应该先声明后引用。如果程序中对一个函数的调用位于该函数的定义之前,则应该在调用前对被调用函数进行声明。函数声明定义了函数原型。函数声明的一般形式为:

返回值类型 函数名(参数类型表);

声明函数原型的目的在于告诉编译器传递给函数的参数个数、类型以及函数返回值的类型,参数表中仅需依次列出函数定义中的参数的类型。函数原型可以使编译器检查源程序中对函数的调用是否正确。

3) 函数调用

当需要在一个函数(称为主调函数)中使用另一个函数(称为被调函数)实现的功能时,便以函数名字进行调用,称为函数调用。在使用一个函数时,只要知道如何调用就可以了,不需要关心被调用函数的内部实现。因此,主调函数需要知道被调函数的名字、返回值和需要向被调函数传递的参数(个数、类型和顺序)等信息。

函数调用的一般形式为: 函数名(实参表);

在 C 程序的执行过程中,通过函数调用可以实现函数定义时描述的功能。函数体中若调用自己,则称为递归调用。

C 和 C++ 通过传值方式将实参传递给形参。调用函数和被调用函数之间交换信息的方法主要有两种:一种是由被调用函数把返回值返回给主调函数,另一种是通过参数带回信息。函数调用时实参与形参间交换信息的方法有传值调用和引用调用两种。

(1) 传值调用。

若函数调用时以实参向形式参数传递相应类型的值,则称为传值调用。在这种方式下,形式参数不能向实际参数返回信息。

例如,定义一个交换两个整型变量值的函数 swap()。

```
void swap(int x,int y)    {           /* 要求调用该函数时传递两个整型的值 */
    int temp;
    temp=x;  x=y;  y=temp;
}
```

函数调用: swap(a,b);

因为是传值调用,swap 函数运行后只能交换 x 和 y 的值,而实际参数 a 和 b 的值却没有发生变化。

在 C 语言中,要实现被调用函数对实际参数的修改,必须用指针作形参。即调用时需先对实参进行取地址运算,然后将实参的地址传递给指针形参(本质上仍属于传值调用)。

例如,定义一个交换两个整型变量的值的函数 swap()。

```
void swap(int *px, int *py) {    /* 交换 *px 和 *py */
    int temp;
    temp = *px;    *px = *py;    *py = temp;
}
```

函数调用: swap(&a, &b);

因为形式参数 px、py 分别得到了变量 a、b 的地址,所以 px 指向的对象 *px 即为 a,py 指向的对象 *py 就是 b,所以在函数中交换 *px 和 *py 的值实际上就是交换实参 a 和 b 的值,从而实现了主调函数中两个整型变量值的交换。这种方式实现了间接内存访问。

(2) 引用调用。

引用是 C++ 中增加的数据类型,当形式参数为引用类型时,函数中对形参的访问和修改实际上就是针对相应的实际参数所作的访问和改变。例如:

```
void swap(int &x, int &y)    {    /* 交换 x 和 y */
    int temp;
    temp = x;    x = y;    y = temp;
}
```

函数调用: swap(a, b);

上述引用调用将实际参数的地址传递给形式参数,使得形参的地址就是对应的实参的地址。在调用 swap(a, b) 中, x、y 就是 a、b 的别名。因此,函数调用完成后,即交换了 a 和 b 的值。

4.2 语言处理程序基础

语言处理程序是一类系统软件的总称,其主要作用是将高级语言或汇编语言编写的程序翻译成某种机器语言程序,使程序可在计算机上运行。语言处理程序主要分为汇编程序、编译程序和解释程序等 3 种基本类型。

4.2.1 汇编程序基本原理

1. 汇编语言

汇编语言是为特定的计算机或计算机系统设计的面向机器的符号化程序设计语言。用汇编语言编写的程序称为汇编语言源程序。因为计算机不能直接识别和运行符号语言程序,所以要用专门的翻译程序——汇编程序进行翻译。用汇编语言编写程序时要遵循所用语言的规范和约定。

汇编语言源程序由若干条语句组成。一个程序中可以有 3 类语句:指令语句、伪指令语句和宏指令语句。

1) 指令语句

指令语句又称为机器指令语句,将其汇编后能产生相应的机器代码,这些代码能被 CPU 直接识别并执行相应的操作。基本的指令有 ADD、SUB 和 AND 等。书写指令语句时必须遵循指令的格式要求。

指令语句可分为传送指令、算术运算指令、逻辑运算指令、移位指令、转移指令和处理机控制指令等。

2) 伪指令语句

伪指令语句指示汇编程序在汇编源程序时完成某些工作,比如给变量分配存储单元地址,给某个符号赋一个值等。伪指令语句与指令语句的区别是:伪指令语句经汇编后不产生机器代码,而指令语句经汇编后要产生相应的机器代码;另外,伪指令语句所指示的操作是在源程序被汇编时完成的,而指令语句的操作必须在程序运行时完成。

通常,汇编语言都应设立下面一些伪指令。

(1) 常数定义伪指令语句:例如,IBM360/370 汇编语言中定义常数语句的格式如下:

```
名字 DC x, y, ...
```

其中,DC 是语句的记忆码,x,y 等规定了用户定义的常数。汇编程序会把所定义的常数按规定依次装入以名字为起始地址的一系列存储单元中。

(2) 存储定义伪指令语句:IBM360/370 的汇编语言中用 DS 语句来定义内存单元,例如:

```
TAB1 DS 100C
```

其中,100 表示存储区域能存储的字节数,TAB1 代表被分配的存储区域的首地址。

(3) 开始伪指令语句:开始语句用来指出源程序段的开头。

(4) 结束伪指令语句:结束语句用来指出源程序段的结束。

每一条汇编指令语句被划分成 4 个区,依次是标号区、操作码区、操作数区和注解区,各个区域之间用确定的符号分隔开。标号区中的标号用于指示一条汇编指令语句,它实际上代表该指令的内存单元地址。操作码区是该语句的指令助记符,它可以是机器指令助记符、伪指令码等。操作数区指出本条汇编指令所操作的运算对象,用寻址方式指定操作数的来源,常用的是寄存器操作数和内存单元操作数。

3) 宏指令语句

在汇编语言中,还允许用户将多次重复使用的程序段定义为宏。宏的定义必须按照相应的规定进行,每个宏都有相应的宏名。在程序的任意位置,若需要使用这段程序,只要在相应的位置使用宏名,就相当于使用了这段程序。因此,宏指令语句就是宏的引用。

2. 汇编程序

汇编程序的功能是将用汇编语言编写的源程序翻译成机器指令程序。汇编程序的基本工作包括:将每一条可执行汇编语句转换成对应的机器指令;处理源程序中出现的伪指令和宏指令。由于汇编指令中形成操作数地址的部分可能出现后面才会有定义的符号,所以汇编程序一般至少需要两次扫描源程序才能完成翻译过程。

第一次扫描的主要工作是定义符号的值并创建一个符号表 ST。ST 记录了汇编时所遇到的符号的值。另外有一个固定的机器指令表 MOT1,其中记录了每条机器指令的记亿码和指令的长度。在汇编程序翻译源程序的过程中,为了计算各汇编语句中标号的地址,需要设立一个位置计数器或单元地址计数器 LC(location counter),其初值一般为 0。在扫描源程序时,每处理完一条机器指令或一条与存储分配有关的伪指令(如定义常数语句、定义存储语句),LC 的值就增加相应的长度。这样,在汇编过程中,LC 的内容就是下一条被汇编的指令的偏移地址。若正在汇编的语句有标号,则该标号的值就取 LC 的当前值。

此外,在第一次扫描中,还需要对与定义符号值有关的伪指令进行处理。为了叙述方便,不妨设立伪指令表 POT1。POT1 表的每一个元素只有两个域:伪指令助记符和相应的子程序入口。下面的步骤①~⑤描述了汇编程序第一次扫描源程序的过程。

- ① 单元计数器 LC 置初值 0;
- ② 打开源程序文件;
- ③ 从源程序中读入第一条语句;
- ④ while (当前语句不是结束语句)

```
{  
    if(当前语句有标号)则将标号和单元计数器 LC 的当前值填入符号表 ST;  
    if(当前语句是可执行的汇编指令语句)则查找 MOT1 表,获得当前指令的长度 K,并令 LC =  
    LC+K;
```

if(当前指令是伪指令)则查找 POT1 表并调用相应的子程序;

if(当前指令的操作码是非法记忆码)则调用出错处理子程序;

从源程序中读入下一条语句;

}

⑤ 关闭源程序文件。

第二次扫描的任务是产生目标程序。除了使用前一次扫描所生成的符号表 ST 外,还要使用机器指令表 MOT2。MOT2 表中的元素有机器指令助记符,机器指令的二进制操作码(binary-code),以及格式(type)和长度(length)等。此外,还要设立一个伪指令表 POT2,供第二次扫描时使用。POT2 的每一元素仍有两个域:伪指令记忆码和相应的子程序入口。与第一次扫描的不同之处是:在第二次扫描中,伪指令有着完全不同的处理。

在第二次扫描中,可执行汇编语句应被翻译成对应的二进制代码机器指令。这一过程涉及两个方面的工作:一是把机器指令助记符转换成二进制机器指令操作码,这可通过在 MOT2 中执行查找操作来实现;二是求出操作数区各操作数的值(用二进制表示)。在此基础上,可以装配出用二进制代码表示的机器指令。从求值的角度看,第二部分工作并不复杂。由于形成操作数地址的各个部分都以表达式形式出现,只要定义一个过程 eval-expr(index,value)即可,其功能是通过 index 给定一个表达式在汇编语句缓冲区 S 中的开始位置,该过程就用 value 返回此表达式的值。

我们可以写出处理这种指令的程序段(假定 index 已指向操作数在缓冲区 S 中的首地址)如下:

```
eval-expr(index,R1);
index := index + 1;
eval-expr(index,N2);
if S[index] = ',' then
begin
    index := index + 1;
    eval-expr(index,X2);
end
else
    X2 := 0;
```

类似地可以写出其他类型的指令处理操作数的程序段。设当前可执行汇编语句的操作助记符在 MOT2 表中的索引值为 i,则整个可执行汇编语句的处理过程可概要描述如下:


```

OP := MOT2[i]. binary-code;
TYPE := MOT2[i]. type;
case TYPE of
    'X': 求 X 型指令操作数各部分的值, 然后按规定字节形成指令;
    ...
end;
将形成的指令送往输出区;

```

在第二次扫描中, 根据伪指令助记符, 调用 POT2 表中相应元素所规定的子程序处理伪指令。DS 伪指令的主要目的是预留存储空间。不妨设一个工作单元 K, 用以累计以字节为单位的存储空间大小, 初值为 0。从 DS 伪指令的操作数区求出 K 的大小后, 就向输出区送 K 个空格以达到保留所规定存储单元的目的。DC 伪指令处理的结果是向输出区送出经转换得到的常量。最后, 开始伪指令工作是输出目标程序开始的标准信息, 而结束伪指令则是输出目标程序结束的标准信息, 这些信息都是为装配程序提供的。

4.2.2 编译程序基本原理

1. 编译过程概述

编译程序的功能是把用高级语言书写的源程序翻译成与之等价的目标程序(汇编语言程序或机器语言程序)。编译程序的工作过程可以分为 6 个阶段, 如图 4-4 所示。实际的编译器中可能会将其中的某些阶段结合在一起进行处理。下面简要介绍各阶段实现的主要功能。

1) 词法分析阶段

源程序可以简单地看作一个多行的字符串。词法分析阶段是编译过程的第一阶段, 这个阶段的任务是对源程序从前到后(从左到右)逐个字符进行扫描, 从中识别出一个个“单词”符号。“单词”符号是程序设计语言的基本语法单位, 如关键字(或称保留字)、标识符、常数、运算符和分隔符(标点符号、左右括号等)。词法分析程序输出的“单词”采用二元组的方式, 即单词类别和单词自身的值。

词法分析过程依据的是语言的语法规则, 即描述“单词”结构的规则。例如, 某 Pascal 源程序中的一条声明语句和赋值语句:

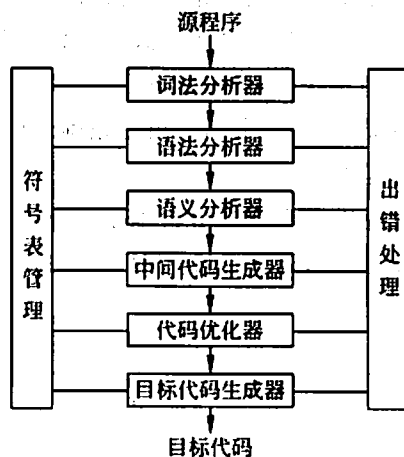


图 4-4 编译器的工作阶段示意图

```
VAR X,Y,Z: real;
X := Y+Z * 60;
```

词法分析阶段将把构成这两条语句的字符串分割成如下的单词序列：

(1) 保留字	VAR	(2) 标识符	X
(3) 逗号	,	(4) 标识符	Y
(5) 逗号	,	(6) 标识符	Z
(7) 冒号	:	(8) 标准标识符	real
(9) 分号	;	(10) 标识符	X
(11) 赋值号	:=	(12) 标识符	Y
(13) 加号	+	(14) 标识符	Z
(15) 乘号	*	(16) 整常数	60
(17) 分号	;		

对于标识符 X、Y 和 Z，其单词类别都是 id(标识符)，字符串“X”、“Y”和“Z”都是单词的值。而对于单词 60，整常数是该单词的类别，60 是该单词的值。这里用 id1、id2 和 id3 分别代表 X、Y 和 Z，强调标识符的内部标识由于组成该标识符的字符串不同而有所区别。经过词法分析后，声明语句“VAR X,Y,Z: real;”表示为“VAR id1,id2,id3:real;”，赋值语句“X := Y+Z * 60;”表示为“id1 := id2+id3 * 60;”。

2) 语法分析阶段

语法分析的任务是在词法分析的基础上，根据语言的语法规则将单词符号序列分解成各类语法单位，如“表达式”、“语句”和“程序”等。语法规则就是各类语法单位的构成规则。通过语法分析，确定整个输入串是否构成一个语法上正确的程序。如果源程序中没有语法错误，语法分析后就能正确地构造出其语法树。否则就指出语法错误，并给出相应的诊断信息。

词法分析和语法分析本质上都是对源程序的结构进行分析。

对语句“id1 := id2+id1 * 60”进行语法分析后形成的语法树如图 4-5 所示。

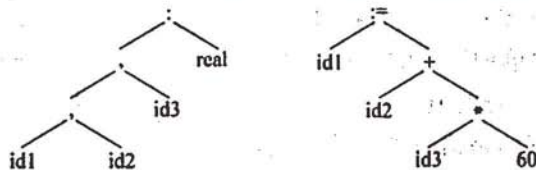


图 4-5 语法树示意图

3) 语义分析阶段

语义分析阶段主要检查源程序是否存在语义错误,并收集类型信息供后面的代码生成阶段使用。只有语法和语义都正确的源程序才能翻译成正确的目标代码。

语义分析的一个主要工作是进行类型分析和检查。程序语言中的数据类型一般包含两个方面的内容:类型的载体及其上的运算。例如,整除取余只能对整型数据进行运算,若其运算对象中有浮点数就认为出现了类型不匹配的错误。

在确认源程序的语法和语义正确之后,就可对其进行翻译,同时改变源程序的内部表示。对于声明语句,需要记录所遇到的符号的信息,所以应对符号表进行填查操作。在图 4-6 所示的符号表中,每一行存放一个符号的信息。第一行存放标识符 X 的信息,其类型为 real,为它分配的地址是 0。第二行存放 y 的信息,它的类型是 real,为它分配的地址是 4。以上信息表明,在这种语言中,为一个 real 型数据分配的存储空间是 4 个存储单元。对于可执行语句,则检查结构合理的表达式是否有意义。对“ $\text{id1} := \text{id2} + \text{id3} * 60$ ”进行语义分析后的语法树如图 4-6 所示,其中增加了一个语义处理节点 `inttoreal`,该运算用于将一个整型数转换为浮点数。

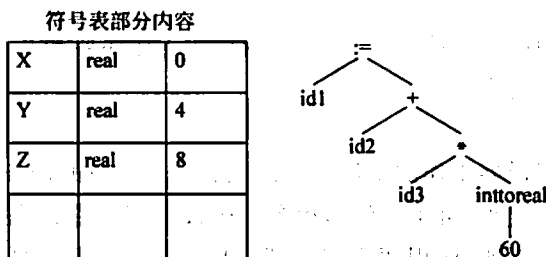


图 4-6 语义分析后的符号表和语法树示意图

4) 中间代码生成阶段

中间代码生成阶段的工作是根据语义分析的输出生成中间代码。“中间代码”是一种简单且含义明确的记号系统,可以有若干种形式,它们的共同特征是与具体的机器无关。中间代码的设计原则主要有两点:一是容易生成;二是容易被翻译成目标代码。最常用的一种中间代码是与汇编语言的指令非常相似的三地址码,其实现方式常采用四元式。四元式的形式为(运算符,运算对象 1,运算对象 2,运算结果)。

例如对语句“ $X := Y + Z * 60$ ”,可生成以下四元式序列:

(1) (`inttoreal`, 60, -, `t1`)

(2) (`*`, `id3`, `t1`, `t2`)

(3) (`+`, `id2`, `t2`, `t3`)

(4) ($t_1 := t_3 - id_1$)

其中 $t_i (i=1,2,3)$ 是编译程序生成的临时变量,用于存放临时的运算结果。

语义分析和中间代码生成所依据的是语言的语义规则。

5) 代码优化阶段

优化是一个编译器的重要组成部分。由于编译器将源程序翻译成中间代码的工作是机械的、按固定模式进行的,因此,生成的中间代码往往在时间上和空间上有很大的浪费。当需要生成高效的目标代码时,就必须进行优化。优化过程可以在中间代码生成阶段进行,也可以在目标代码生成阶段进行。由于中间代码是不依赖于具体机器的,此时所作的优化一般建立在对程序的控制流和数据流分析的基础之上,与具体的机器无关。优化所依据的原则是程序的等价变换规则。例如,在生成“ $X := Y + Z * 60$ ”的四元式后,60 是编译时已知的常数,把它转换为 60.0 的工作可以在编译时完成,没有必要生成一个四元式,同时 t_3 仅仅用来将其值传递给 id_1 ,也可以被化简掉,因此上述的中间代码可优化成下面的等价代码:

① ($*$, $id_3, 60.0, t_1$)

② ($+$, id_2, t_1, id_1)

这只是优化工作中的一个简单示例,真正的优化工作还要涉及公共子表达式的提取,以及循环优化等更多的内容。

6) 目标代码生成阶段

目标代码生成是编译器工作的最后一个阶段。这一阶段的任务是把中间代码变换成特定机器上的绝对指令代码、可重定位的指令代码或汇编指令代码。这个阶段的工作与具体的机器密切相关。例如,使用两个寄存器 R_1 和 R_2 ,上述的四元式可生成下面的目标代码:

① MOVF id_3, R_2

② MULF $\# 60.0, R_2$

③ MOVF id_2, R_1

④ ADDF R_2, R_1

⑤ MOV R_1, id_1

这里用 $\#$ 表明 60.0 为常数。

7) 符号表管理

符号表的作用是记录源程序中各个符号的必要信息,以辅助语义的正确性检查和代码生成。在编译过程中需要对符号表进行快速有效地查找、插入、修改和删除等操作。符号表的建立可以始于词法分析阶段,也可以放到语法分析和语义分析阶段,但符号表的使用有时会延续到目标代码的运行阶段。

8) 出错处理

用户编写的源程序不可避免地会有一些错误,这些错误大致可分为静态错误和动态错误。动态错误也称动态语义错误,指程序中包含的逻辑错误。这种错误发生在程序运行时,例如,变量取零时作除数和引用数组元素下标越界等错误。静态错误是指编译时所发现的程序错误。可分为语法错误和静态语义错误,如单词拼写错误、标点符号错、表达式中缺少操作数以及括号不匹配等有关语言结构上的错误称为语法错误,而语义分析时发现的运算符与运算对象类型的不匹配等错误属于静态语义错误。

在编译时发现程序中的错误后,编译程序应采用适当的策略跳过它们,使得分析过程能够继续下去,以便在一次编译过程中尽可能多地找出程序中的错误。

编译器的各个阶段逻辑上可以划分为前端和后端两部分。前端包括从词法分析到中间代码生成各阶段的工作,后端包括中间代码优化、目标代码的生成与优化等阶段。这样,以中间代码为分水岭,把编译器分成了与机器有关的部分和与机器无关的部分。如此一来,对于同一种程序语言的编译器,在开发出一个前端之后,就可以针对不同的机器开发相应的后端,前后端有机结合后就形成了该语言的一个编译器。当语言有改动时,只会涉及前端部分的维护。对于不同的程序语言,分别设计出相应的前端,然后将各个语言的前端与同一个后端相结合,就可得到各个语言在某种机器上的编译器。

2. 文法和语言的形式描述

1) 字母表、字符串、字符串集合及运算

- 字母表 Σ : 元素的非空有穷集合,如 $\Sigma = \{a, b\}$ 。
- 字符: 字母表 Σ 中的一个元素,如 Σ 上的 a 或 b 。
- 字符串: 字母表 Σ 中字符组成的有穷序列。例如, a, ab 和 aaa 都是 Σ 上的字符串。
- 字符串的长度: 指字符串中的字符个数,如 $|aba| = 3$ 。
- 空串 ϵ : 由零个字符组成的序列, $|\epsilon| = 0$ 。
- 连接: 字符串 S 和 T 的连接是指将串 T 接续在串 S 之后,表示为 $S \cdot T$,连接符号“ \cdot ”可省略。显然,对于字母表 Σ 上的任意字符串 $S, S \cdot \epsilon = \epsilon \cdot S = S$ 。
- 空集: 用符号 \emptyset 表示。
- Σ^* : 指包括空串 ϵ 在内的 Σ 上所有字符串的集合,例如,设 $\Sigma = \{a, b\}, \Sigma^* = \{\epsilon, a, b, aa, bb, ab, ba, aaa, \dots\}$ 。
- 字符串的方幂: 把字符串 α 自身连接 n 次得到的串,称为字符串 α 的 n 次方幂,记为 α^n 。 $\alpha^0 = \epsilon, \alpha^n = \alpha \alpha^{n-1} = \alpha^{n-1} \alpha (n > 0)$ 。
- 字符串集合的运算: 设 A, B 代表字母表 Σ 上的两个字符串集合。
或(合并): $A \cup B = \{\alpha | \alpha \in A \text{ 或 } \alpha \in B\}$ 。

积(连接): $AB = \{\alpha\beta | \alpha \in A \text{ 且 } \beta \in B\}$ 。

幂: $A^0 = A \cdot A^{n-1} = A^{n-1} \cdot A (n > 0)$, 并规定 $A^0 = \{\epsilon\}$ 。

正则闭包+: $A^+ = A^1 \cup A^2 \cup \dots \cup A^n \cup \dots$ 。

闭包*: $A^* = A^0 \cup A^+$ 。显然, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n \cup \dots$ 。

2) 文法和语言的形式描述

语言 L 是有限字母表 Σ 上有限长度字符串的集合, 这个集合中的每个符号串都是按照一定的规则生成的。下面从产生语言的角度出发, 给出文法和语言的形式定义。所谓产生语言, 是指制定出有限条规则, 借助它们就能产生此语言的全部句子。

(1) 文法的定义: 描述语言语法结构的形式规则称为文法。文法 G 是一个四元组, 可表示为: $G = (V_N, V_T, P, S)$, 其中 V_T 是一个非空有限集, 其中的每个元素称为一个终结符; V_N 是一个非空有限集, 其中的每个元素称为非终结符。 $V_N \cap V_T = \emptyset$, 即 V_N 和 V_T 不含公共元素。令 $V = V_N \cup V_T$, 称 V 为文法 G 的词汇表, V 中的符号称为文法符号, 包括终结符和非终结符。 $S \in V_N$, 称为开始符号, 它至少要在一条产生式中作为左部出现。 P 是产生式的有限集合, 每个产生式是形如下列给出的规则:

$$\alpha \rightarrow \beta$$

其中, α 称为产生式的左部, $\alpha \in V^+$ 且 α 中至少含有一个非终结符; β 称为产生式的右部且 $\beta \in V^*$ 。若干个产生式 $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$ 的左部相同时, 可简写为 $\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$; 称 $\beta_i (1 \leq i \leq n)$ 为 α 的一个候选式。

(2) 文法的分类: 乔姆斯基(Chomsky)把文法分成 4 种类型, 即 0 型、1 型、2 型和 3 型。这 4 类文法之间的差别在于对产生式施加不同的限制。若文法 $G = (V_N, V_T, P, S)$ 的每个产生式 $\alpha \rightarrow \beta$, 均有 $\alpha \in (V_N \cup V_T)^*$, α 至少含有一个非终结符, 且 $\beta \in (V_N \cup V_T)^*$, 则称 G 为 0 型文法。对 0 型文法的每条产生式分别施加以下的第 i 条限制, 则可得 i 型文法:

- G 的任何产生式 $\alpha \rightarrow \beta (S \rightarrow \epsilon \text{ 除外})$ 均满足 $|\alpha| \leq |\beta|$ ($|x|$ 表示 x 中文法符号的个数);
- G 的任何产生式形如 $A \rightarrow \beta$, 其中 $A \in V_N, \beta \in (V_N \cup V_T)^*$;
- G 的任何产生式形如 $A \rightarrow a$ 或 $A \rightarrow aB$ (或者 $A \rightarrow Ba$), 其中 $A, B \in V_N, a \in V_T$ 。

0 型文法也称为短语文法, 其能力相当于图灵机。任何 0 型语言都是递归可枚举的; 反之, 递归可枚举集也必定是一个 0 型语言。1 型文法也称为上下文有关文法, 这种文法意味着对非终结符的替换必须考虑上下文, 并且一般不允许替换成 ϵ 串。例如, 若 $\alpha A \beta \rightarrow \alpha \gamma \beta$ 是 1 型文法的产生式, α 和 β 不全为空, 则非终结符 A 只有在左边是 α , 右边是 β 的上下文中才能替换成 γ 。2 型文法是上下文无关文法, 对非终结符的替换无须考虑上下文。3 型文法等价于正规式, 因此也称为正规文法或线性文法。

(3) 句子和语言: 设有文法 $G = (V_N, V_T, P, S)$, 下面引入一些概念。

- 推导与直接推导: 推导就是从文法的开始符号 S 出发, 反复使用产生式, 将产生式左部的非终结符替换为右部的文法符号序列(展开产生式用 \Rightarrow 表示), 直至产生一个终结符的序列时为止。若有产生式 $\alpha \rightarrow \beta \in P, \gamma, \delta \in V^*$, 则 $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ 称为文法 G 中的一个直接推导, 并称 $\gamma\alpha\delta$ 可直接推导出 $\gamma\beta\delta$ 。显然, 对 P 中的每一个产生式 $\alpha \rightarrow \beta$, 都有 $\alpha \Rightarrow \beta$ 。若在文法中存在一个直接推导序列, 即 $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n (n > 0)$, 则称 α_0 可推导出 α_n , α_n 是 α_0 的一个推导, 并记为 $\alpha_0 \xRightarrow{+}_G \alpha_n$ 。用记号 $\alpha_0 \xRightarrow{*}_G \alpha_n$ 表示 $\alpha_0 = \alpha_n$ 或者 $\alpha_0 \xRightarrow{+}_G \alpha_n$ 。
- 直接归约和归约(推导的逆过程): 若文法 G 中有一个直接推导 $\alpha \Rightarrow \beta$, 则称 β 可直接归约成 α , 或 α 是 β 的一个直接归约。若文法 G 中有一个推导 $\gamma \xRightarrow{*}_G \delta$, 则称 δ 可归约成 γ , 或 γ 是 δ 的一个归约。
- 句型 and 句子: 若文法 G 的开始符号为 S , 那么, 从开始符号 S 能推导出的符号串称为文法的一个句型。即 α 是文法 G 的一个句型, 当且仅当有如下推导 $S \xRightarrow{*}_G \alpha, \alpha \in V^*$ 。若 X 是文法 G 的一个句型, 且 $X \in V_T^+$, 则称 X 是文法 G 的一个句子。即仅含终结符的句型是一个句子。
- 语言: 从文法 G 的开始符号出发, 所能推导出的句子的全体称为文法 G 产生的语言, 记为 $L(G)$ 。它可以表示成: $L(G) = \{X | S \xRightarrow{*}_G X, X \in V_T^+\}$ 。

(4) 文法的等价: 若文法 G_1 与文法 G_2 产生的语言是相同的, 即 $L(G_1) = L(G_2)$, 则称这两个文法是等价的。

在形式语言理论和编译理论中, 文法等价是一个很重要的概念, 根据这一概念, 可对文法进行等价改造, 以得到所需形式的文法。

3. 词法分析

语言中具有独立含义的最小语法单位是符号(单词), 如标识符、无符号常数与界限符等。词法分析的任务是把构成源程序的字符串转换成单词符号序列。词法规则可利用 3 型文法(正规文法)或正规表达式描述。它产生的集合是语言基本字符集 Σ (字母表)上的字符串的一个子集, 我们称为正规集。

1) 正规表达式和正规集

对于字母表 Σ , 其正规式及其表示的正规集可以递归定义如下:

- (1) ϵ 是一个正规式, 它表示集合 $L(\epsilon) = \{\epsilon\}$;
- (2) 若 a 是 Σ 上的字符, 则 a 是一个正规式, 它所表示的正规集为 $\{a\}$;

(3) 若正规式 r 和 s 分别表示正规集 $L(r)$ 和 $L(s)$, 则

- $r|s$ 是正规式, 表示集合 $L(r) \cup L(s)$;
- $r \cdot s$ 是正规式, 表示集合 $L(r)L(s)$;
- r^* 是正规式, 表示集合 $(L(r))^*$;
- (r) 是正规式, 表示集合 $L(r)$ 。

仅由有限次地使用上述三个步骤定义的表达式才是 Σ 上的正规式。

运算符“|”、“ \cdot ”和“ $*$ ”分别称为“或”、“连接”和“闭包”。在正规式的书写中, 连接运算符“ \cdot ”可省略。运算的优先级从高到低顺序排列依次为: “ $*$ ”、“ \cdot ”和“|”。

设 $\Sigma = \{a, b\}$, 表 4-1 列出 Σ 上的一些正规式和相应的正规集。

表 4-1 正规式与正规集

正规式	正规集
ab	符号串 ab 构成的集合
$a b$	符号串 a, b 构成的集合
a^*	由零个或多个 a 构成的符号串集合
$(a b)^*$	所有由字符 a 和 b 构成的串的集合, 包括空串 ϵ
$a(a b)^*$	以 a 为首字符的 a, b 字符串的集合
$(a b)^*abb$	以 abb 结尾的 a, b 字符串的集合

若两个正规式表示的正规集相同, 则认为二者等价。两个等价的正规式 U 和 V 记为 $U=V$ 。例如, $b(ab)^* = (ba)^*b$, $(a|b)^* = (a^*b^*)^*$ 。设 U, V 和 W 均为正规式, 正规式的代数性质如表 4-2 所示。

表 4-2 正规式的代数性质

公理	公理
$U V=V U$	$(UV)W=U(VW)$
$(U V)W=U(VW)$	$\epsilon U=U\epsilon=U$
$U(V W)=UV UW$	$V^*=(V^+ \epsilon)$
$(U V)W=UW VW$	$V^{**}=V^*$

2) 有限自动机

有限自动机是一种识别装置的抽象概念, 它能准确地识别正规集。基本的有限自动机分为两类: 确定的有限自动机和不确定有限自动机。

(1) 确定的有限自动机(deterministic finite automata, DFA)。

一个确定的有限自动机(DFA)是一个五元组: (S, Σ, f, s_0, Z) , 其中:

- S 是一个有限集合, 它的每个元素称为一个状态;
- Σ 是一个有穷字母表, 它的每个元素称为一个输入字符;
- f 是 $S \times \Sigma \rightarrow S$ 上的单值部分映像。 $f(A, a) = Q$ 表示当前状态为 A 、输入为 a 时, 将转换到下一状态 Q 。我们称 Q 为 A 的一个后继状态;
- $s_0 \in S$, 是惟一的一个开始状态;
- Z 是非空的终止状态集合, $Z \subseteq S$ 。

一个 DFA 可以用两种直观的方式表示: 状态转换图和状态转换矩阵。状态转换图简称为转换图, 它是一个有向图。DFA 中的每个状态对应转换图中的一个节点。DFA 中的每个转换函数对应图中的一条有向弧。若转换函数为 $f(A, a) = Q$, 该有向弧从节点 A 出发, 进入节点 Q , 字符 a 是弧上的标记。

【例 4.1】 已知 DFA $M1 = (\{s_0, s_1, s_2, s_3\}, \{a, b\}, f, S, \{s_3\})$, 其中 f 为:

$$\begin{aligned} f(s_0, a) = s_1, \quad f(s_0, b) = s_2, \quad f(s_1, a) = s_3, \quad f(s_1, b) = s_2, \\ f(s_2, a) = s_1, \quad f(s_2, b) = s_3, \quad f(s_3, a) = s_3 \end{aligned}$$

与 DFA $M1$ 对应的状态转换图如图 4-7(a) 所示, 其中, 双圈表示的节点是终态节点。状态转换矩阵可以用一个二维数组 M 表示, 矩阵元素 $M[A, a]$ 的行下标表示状态, 列下标表示输入字符, $M[A, a]$ 的值是当前状态为 A 、输入字符为 a 时, 应转换到的下一状态。状态转换矩阵如图 4-7(b) 所示。在转换矩阵中, 一般以第一行的行下标对应的状态作为初态, 而终态则需要特别指出。

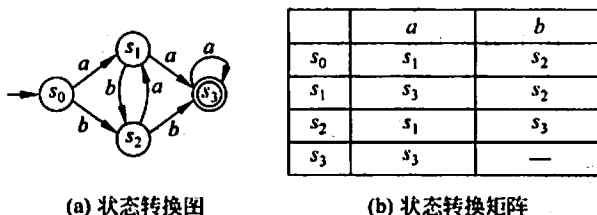


图 4-7 确定的有限自动机示意图

对于 Σ 中的任何字符串 ω , 若存在一条从初态节点到某一终状态节点的路径, 且这条路径上所有弧的标记符连接成的字符串等于 ω , 则称 ω 可由 DFA M 识别(接受或读出)。若一个 DFA M 的初态节点同时又是终态节点, 则空字 ϵ 可由该 DFA M 识别(或接受)。 M 所能识别的语言 $L(M) = \{\omega | \omega \text{ 是从 } M \text{ 的初态到终态的路径上的弧上标记符所形成的串}\}$ 。

Σ 上的一个字符串集合 V 是正规的, 当且仅当存在 Σ 上的一个确定有限自动机 M , 且 $V = L(M)$ 。

(2) 不确定的有限自动机(nondeterministic finite automata, NFA)。

一个不确定的有限自动机也是一个五元组,它与确定有限自动机的区别是:

- f 是 $S \times \Sigma \rightarrow 2^S$ 上的映像。对于 S 中的一个给定状态及输入符号,返回一个状态的集合。即当前状态的后继状态不一定是惟一的。
- 有向弧上的标记可以是 ϵ 。

显然,DFA 是 NFA 的特例。实际上,对于每个 NFA M ,都存在一个 DFA N ,且 $L(M)=L(N)$ 。

对于任何两个有限自动机 $M1$ 和 $M2$,如果 $L(M1)=L(M2)$,则称 $M1$ 和 $M2$ 是等价的。

3) NFA 到 DFA 的转换

(1) 若 I 是 NFA N 的状态集合的一个子集,我们定义 $\epsilon_CLOSURE(I)$ 如下:

- 状态集 I 的 $\epsilon_CLOSURE(I)$ 是一个状态集;
- 状态集 I 的所有状态属于 $\epsilon_CLOSURE(I)$;
- 若 $s \in I$,那么从 s 出发经过任意条 ϵ 弧到达的状态 s' 都属于 $\epsilon_CLOSURE(I)$ 。

状态集 $\epsilon_CLOSURE(I)$ 称为 I 的 ϵ -闭包。

由以上的定义可知, I 的 ϵ -闭包就是从状态集 I 的状态出发,经 ϵ 弧所能到达的状态的全体。

假定 I 是 N 的状态集的一个子集, a 是 Σ 中的一个字符,我们定义:

$$I_a = \epsilon_CLOSURE(J)$$

其中, J 是所有那些可从 I 中的某一状态节点出发经过一条 a 弧而到达的状态节点的全体。

在定义了 $\epsilon_CLOSURE$ 后,我们就可以用子集法将一个非确定的有限自动机转化为一个确定的有限自动机。

(2) NFA 转换为 DFA,设 NFA $N=(S, \Sigma, f, s_0, Z)$,与之等价的 DFA $M=(S', \Sigma, f', q_0, Z')$,用子集法将非确定的有限自动机确定化的算法步骤如下:

① 求出 DFA M 的初态 q_0 ,即令 $q_0 = \epsilon_CLOSURE(\{s_0\})$ 。此时 S' 仅含初态 q_0 ,并且没有标记;

② 对于 S' 中尚未标记的状态 $q_i = \{s_{i1}, s_{i2}, \dots, s_{im}\}$ 和 $s_{ij} \in S (j=1, \dots, m)$ 进行以下处理:

- 标记 q_i ;
- 对于每个 $a \in \Sigma$,令 $T = f(\{s_{i1}, s_{i2}, \dots, s_{im}\}, a)$, $q_j = \epsilon_CLOSURE(T)$;
- 若 q_j 尚不在 S' 中,则将 q_j 作为一个未加标记的新状态添加到 S' ,并把状态转换函数 $f'(q_i, a) = q_j$ 添加到 DFA M 中;

③ 重复步骤②,直到 S' 中不再有未标记的状态时为止;

④ 令 $Z' = \{q | q \in S' \text{ 且 } q \cap Z \neq \emptyset\}$ 。

【例 4.2】已知一个识别正规式 ab^*a 的非确定有限自动机(NFA M),其状态转换图如图 4-8 所示。试用子集法将其转换为 DFA M 。

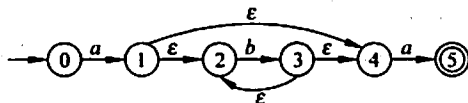


图 4-8 NFA M 的状态转换图

根据 $\epsilon_CLOSURE$ 的定义,可以求出 $\epsilon_CLOSURE(\{0\}) = \{0\}$,将 $\{0\}$ 记为 DFA 的初态 q_0 。然后根据题中所给的状态转换图以及算法步骤②,求解 DFA M 各个状态的过程如下。

$\epsilon_CLOSURE(q_0, a) = \{1, 2, 4\}$,将 $\{1, 2, 4\}$ 记为 q_1 , $\epsilon_CLOSURE(q_0, b) = \{\}$,标记 q_0 。

$\epsilon_CLOSURE(q_1, a) = \{5\}$,将 $\{5\}$ 记为 q_2 (终态)。

$\epsilon_CLOSURE(q_1, b) = \{2, 3, 4\}$,将 $\{2, 3, 4\}$ 记为 q_3 ,标记 q_1 。

$\epsilon_CLOSURE(q_2, a) = \{\}$, $\epsilon_CLOSURE(q_2, b) = \{\}$,标记 q_2 。

$\epsilon_CLOSURE(q_3, a) = \{5\}$,即 q_2 。

$\epsilon_CLOSURE(q_3, b) = \{2, 3, 4\}$,即 q_3 。标记 q_3 。

当 S' 中不再有未标记的状态时,算法即可终止,转换所得的 DFA N 如图 4-9 所示。

(3) 对于 DFA 的最小化,从 NFA 转换得到的 DFA 不一定是简化的,可以通过等价变换将 DFA 作最小化处理。

对于有限自动机中的任何两个状态 t 和 s ,若从其中一个状态出发接受输入字符串 ω ,而从另一状态出发则不接受 ω ,或者从 t 和 s 出发到达不同的接受状态,则称 ω 对状态 s 和 t 是可区分的。若状态 s 和 t 不可区分,则称其为可以合并的等价状态。对图 4-9 所示的自动机进行化简所得的 DFA 如图 4-10 所示。

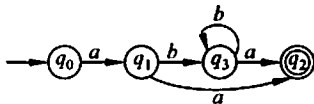


图 4-9 DFA N 的状态转换图

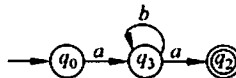


图 4-10 最小化 DFA N 的状态转换图

4) 正规式与有限自动机的转换

(1) 对于 Σ 上的 NFA M ,可以构造一个 Σ 上的正规式 R ,使得 $L(R) = L(M)$ 。

我们把状态转换图的概念拓广,令每条弧用一个正规式作标记,则可为 Σ 上的 NFA M

构造相应的正规式 R 。步骤如下。

① 在 M 的状态转换图中加两个节点：一个 x 节点；一个 y 节点。从 x 节点到 NFA M 的初始状态节点引一条弧并用 ϵ 标记，从 NFA M 的所有终态节点引一条弧到 y 节点并用 ϵ 标记，这样就形成一个与 M 等价的 M' ， M' 只有一个初态 x 和一个终态 y 。

② 按下面的方法逐步消去 M' 中除 x 和 y 的所有节点。在消除节点的过程中，用正规式来标记弧，最后节点 x 和 y 之间的弧上的标记就是所求的正规式。消除节点的规则如图 4-11 所示。

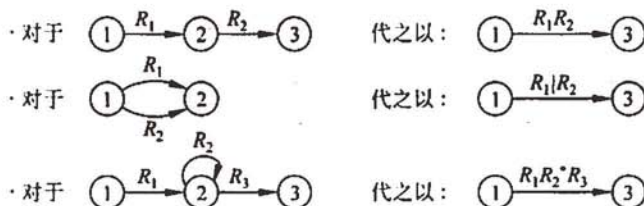


图 4-11 有限自动机到正规式的转换规则示意图

(2) 同样地，对于 Σ 上的每个正规式 R ，可以构造一个 Σ 上的 NFA M ，使得 $L(M) = L(R)$ 。

① 对于正规式 R ，可用图 4-12 所示的拓广状态图表示。

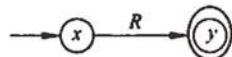


图 4-12 正规式 R 的状态转换图

② 通过对正规式 R 进行分裂并加入新的节点，逐步把状态转换图转变成每条弧上的标记是 Σ 上的一个字符或 ϵ 。转换规则如图 4-13 所示。



图 4-13 正规式到有限自动机的转换规则示意图

最后所得的图即为一个 NFA M ， x 为初态节点， y 为终态节点。显然， $L(M) = L(R)$ 。

5) 词法分析器的构造

有了正规式和有限自动机的理论基础后，就可以构造出编译程序的词法分析模块。构造词法分析器的一般步骤如下：

- ① 用正规式描述语言中的单词构成规则;
- ② 为每个正规式构造一个 NFA, 用于识别正规式所表示的正规集;
- ③ 将构造出的 NFA 转换成等价的 DFA;
- ④ 对 DFA 进行最小化处理, 使其最简;
- ⑤ 根据 DFA 构造词法分析器。

4. 语法分析

语法分析的任务是根据语言的语法规则, 分析单词串是否构成短语和句子, 即表达式、语句和程序等基本语言结构, 同时检查和处理程序中的语法错误。程序设计语言的绝大多数语法规则可以用上下文无关文法描述。语法分析方法有多种, 根据产生语法树的方向, 可分为自底向上和自顶向下两类。

1) 上下文无关文法

上下文无关文法属于乔姆斯基(Chomsky)定义的 2 型文法, 被广泛地用来表示各种程序设计语言的语法规则。对于上下文无关文法 $G[S] = (V_N, V_T, P, S)$, 其产生式的形式都是: $A \rightarrow \beta$, 其中 $A \in V_N, \beta \in (V_N \cup V_T)^*$ 。

若不加特别说明, 下面我们用大写英文字母 A、B、C 等表示非终结符, 小写英文字母 a、b、c 等表示终结符号, u、v、w 等表示终结符号串, 小写希腊字母 $\alpha, \beta, \gamma, \delta$ 等表示终结符和非终结符混合的文法符号串。

(1) 规范推导(最右推导), 如果推导的任何一步 $\alpha \Rightarrow \beta$ (其中 α, β 是句型), 都是对 α 中的最右(最左)非终结符进行替换, 则称这种推导为最右(最左)推导。最右推导常称为规范推导。

(2) 短语、直接短语和句柄: 设 $\alpha\delta\beta$ 是文法 G 的一个句型, 即 $S \Rightarrow^+ \alpha\delta\beta$, 且满足 $S \Rightarrow^+ \alpha A \beta$ 和 $A \Rightarrow^+ \delta$, 则称 δ 是句型 $\alpha\delta\beta$ 相对于非终结符 A 的短语。特别地, 如果有 $A \Rightarrow \delta$, 则称 δ 是句型 $\alpha\delta\beta$ 相对于产生式 $A \rightarrow \delta$ 的直接短语。一个句型的最左直接短语称为该句型的句柄。

【例 4.3】 对于简单算术表达式, 可以用下面的文法进行描述。

$$G[E] = ((\{E, T, F\}, \{+, *, (,), id\}, P, E)$$

$$P = \{E \rightarrow T | E + T, T \rightarrow F | T * F, F \rightarrow (E) | id\}$$

可以证明 $id + id * id$ 是该文法的句子。下面用最右推导从文法的开始符号出发推导出该句子。为了表示推导过程中相同符号的不同出现, 给符号加一个下标。

$$E \Rightarrow E_1 + T_1 \Rightarrow E_1 + T_2 * F_1 \Rightarrow E_1 + T_2 * id_3 \Rightarrow E_1 + F_2 * id_3 \Rightarrow E_1 + id_2 * id_3$$

$$\Rightarrow T_3 + id_2 * id_3 \Rightarrow F_3 + id_2 * id_3 \Rightarrow id_1 + id_2 * id_3$$

对 $id+id*id$ 的推导过程可以用图 4-14 所示的树结构进行描述。

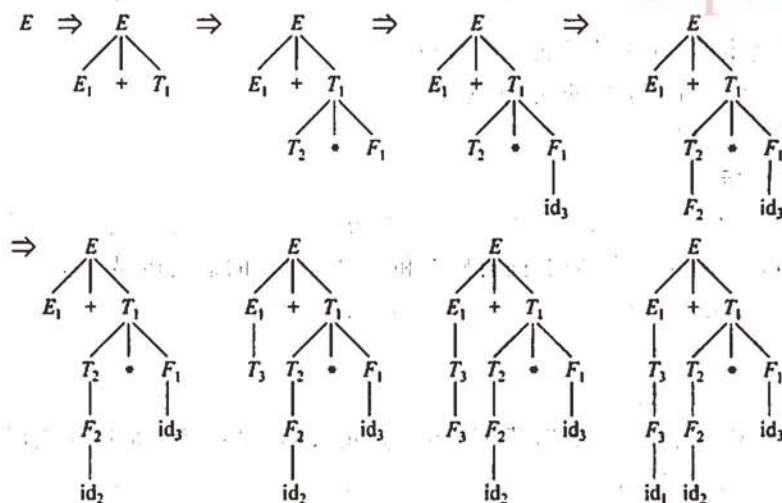


图 4-14 产生句子 $id+id*id$ 的推导过程示意图

由于 $E \Rightarrow F_3 + id_2 * id_3$, 且 $F_3 \Rightarrow id_1$, 所以 id_1 是句型 $id_1 + id_2 * id_3$ 相对于非终结符 F 的短语, 也是相对于产生式 $F \rightarrow id$ 的直接短语。

由于 $E \Rightarrow E_1 + T_2 * id_3$, 且 $T_2 \Rightarrow id_2$, 所以 id_2 是句型 $E_1 + id_2 * id_3$ 相对于非终结符 T 的短语。

由于 $E \Rightarrow E_1 + T_1$, 且 $T_1 \Rightarrow id_2 * id_3$, 所以 $id_2 * id_3$ 是句型 $E_1 + id_2 * id_3$ 相对于非终结符 T 的短语。

由于 $E \Rightarrow id_1 + id_2 * id_3$, 所以 $id_1 + id_2 * id_3$ 是句型 $id_1 + id_2 * id_3$ 相对于非终结符 E 的短语。

实际上 $id_1, id_2, id_3, id_2 * id_3$ 和 $id_1 + id_2 * id_3$ 都是句型 $id_1 + id_2 * id_3$ 的短语, 而且 id_1, id_2, id_3 均是直接短语, 其中 id_1 是最左直接短语, 即句柄。

2) 自顶向下分析方法

自顶向下分析法的基本思想是: 对于给定的输入串 ω , 从文法的开始符号 S 出发进行最左推导, 直至得到一个合法的句子或者发现一个非法结构。在推导的过程中试图用一切可能的方法, 自上而下、从左到右地为输入串 ω 建立语法树。整个分析过程是一个试探的过程, 是反复使用不同产生式谋求与输入序列匹配的过程。若输入串是给定文法的句子, 则必能成功, 反之必然出错。

文法中存在下述产生式时,自顶向下分析过程中会出现下面的问题:

- 若文法中存在形如 $A \rightarrow \alpha\beta | \alpha\delta$ 的产生式,即 A 产生式中有多个候选项的前缀是相同的(称为公共左因子,简称左因子),则可能导致分析过程中的回溯处理;
- 若文法中存在形如 $A \rightarrow A\alpha$ 的产生式,由于采取了最左推导,可能会造成分析过程陷入死循环的情况。产生式的这种形式被称为左递归。

因此,需要对文法进行改造,消除其中的左递归,以避免分析过程陷入死循环。提取左因子,以避免回溯。

(1) LL(1)文法。

非形式地讲,文法符号序列 α 的 FIRST 集合,就是从 α 出发可以推导出的所有以终结符号开头的序列中的开头终结符号。而一个非终结符 A 的 FOLLOW 集合,就是从文法开始符号可以推导出的所有含 A 句型中紧跟在 A 之后的终结符号。

一个文法 G 是 LL(1)的,当且仅当 G 的任何两个产生式 $A \rightarrow \alpha | \beta$ 满足下面的条件:

- 对任何终结符 a , α 和 β 不能同时推导出以 a 开始的文法符号序列;
- α 和 β 最多有一个可以推导出 ϵ ;
- 若 $\beta \Rightarrow \epsilon$,则 α 不能推导出以 FOLLOW(A)中的终结符开始的任何文法符号序列。

(2) 递归下降分析法。

递归下降分析法要求文法是 LL(1)文法,它直接以程序的方式模拟产生式产生语言的过程,其基本思想是:为每一个非终结符构造一个子程序,每个子程序的过程体按该产生式候选项分情况展开,遇到终结符即进行匹配,而遇到非终结符则调用相应的子程序。该分析法从调用文法开始符号的子程序开始,直到所有非终结符都展开为终结符并得到匹配为止。若分析过程可以达到这一步,则表明分析成功,否则表明输入串中有语法错误。对于规模较小的语言,递归下降分析法是一种有效的方法,其优点是简单且易于构造,缺点是程序与文法直接相关,对文法的任何改变都需要在程序中进行相应的修改。

(3) 预测分析法。

预测分析法是另一种自顶向下的分析方法,其基本模型如图 4-15 所示。

一个 LL(1)文法的预测分析表可以用一个二维数组 M 表示,其元素 $M[A, a]$ ($A \in V_N, a \in V_T \cup \#$) 存放关于 A 的产生式,表明当遇到输入符号为 a 且用 A 进行推导时,应采用的产生式。若 $M[A, a]$ 为 error,则表明推导时遇到了不该出现的符号,应进行出错处理。

3) 自底向上分析方法

自底向上的分析方法也称为移进-归约分析法,它的基本思想是对输入序列 ω 自左至

右进行扫描,并将输入符号逐个移进一个栈中,边移进边分析,一旦栈顶符号串形成一个句型的可归约串时,就用某个产生式的左部非终结符来替代,这称为一步归约。重复这一过程,直至栈中只剩下文法的开始符号且输入串也被扫描完为止,此时确认输入串 w 是文法的句子,表明分析成功。否则,进行出错处理。自底向上分析法的工作模型如图 4-16 所示。

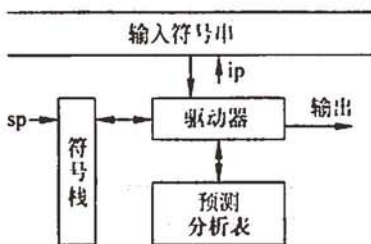


图 4-15 预测分析模型示意图

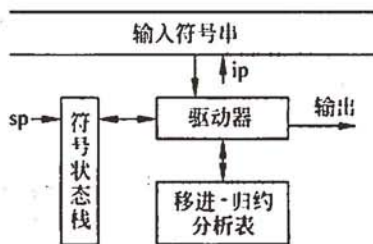


图 4-16 移进-归约分析模型示意图

移进-归约分析法的数学模型也是下推自动机。若模型中采用算符优先分析表,用“最左素短语”来刻画“可归约串”,则相应的分析器就称为算符优先分析器。若采用 LR 分析表,用“句柄”来刻画“可归约串”,则相应的分析器就称为 LR 分析器。下面我们简单介绍 LR 分析器的原理。

LR 分析法是一种规范归约分析法,是规范推导(最右推导)的逆过程。

下面举例说明规范归约的过程。

【例 4.4】 设文法 $G[S] = \{ S \rightarrow aAcBe, A \rightarrow b, A \rightarrow Ab, B \rightarrow d \}$,下面对输入串 $abbcde\#$ 进行分析。先设一个符号栈,并把句子左括号“ $\#$ ”放入栈底,其分析过程如下。

步骤	符号栈	输入符号串	动作
①	#	abbcde#	移进
②	# a	bbcde#	移进
③	# ab	bcde#	归约($A \rightarrow b$)
④	# aA	bcde#	移进
⑤	# aAb	cde#	归约($A \rightarrow Ab$)
⑥	# aA	cde#	移进
⑦	# aAc	de#	移进
⑧	# aAcd	e#	归约($B \rightarrow d$)
⑨	# aAcB	e#	移进
⑩	# aAcBe	#	归约($S \rightarrow aAcBe$)
⑪	# S	#	接受

说明：第③步中栈顶符号串 b 是句型 $abbcde$ 的句柄，用产生式 $A \rightarrow b$ 进行归约；第⑤步中 Ab 是句型 $aAbcde$ 的句柄，用产生式 $A \rightarrow Ab$ 进行归约；第⑧步和第⑩步分别用产生式 $B \rightarrow d$ 和 $S \rightarrow aAcBe$ 进行归约。上述分析过程也可看作自底向上构造语法树的过程。

LR 分析法根据当前分析栈中的符号串（通常以状态表示），同时向右顺序查看输入串的 k 个（ $k \geq 0$ ）符号，就可惟一地确定分析器的动作是移进还是归约，以及用哪条产生式进行归约，因而也就能惟一地确定句柄。当 $k=1$ 时，已能满足当前绝大多数高级语言编译程序的需求。常用的 LR 分析器有 LR(0)、SLR(1)、LALR(1) 和 LR(1) 等。

一个 LR 分析器由 3 个部分组成：

- (1) 驱动器：或称驱动程序，所有 LR 分析器的驱动程序都是相同的。
- (2) 分析表：不同的文法具有不同的分析表。同一文法采用不同的 LR 分析器时，分析表也不同。分析表又可分为动作表 (ACTION) 和状态转换表 (GOTO) 两个部分，它们都可用二维数组表示。
- (3) 分析栈：包括文法符号栈和相应的状态栈。

分析器的动作由栈顶状态和当前输入符号决定 (LR(0) 分析器不需向前查看输入符号)。LR 分析器的模型如图 4-17 所示，其中 SP 为栈顶指针， S_i 为状态， X_i 为文法符号。ACTION[S_i, a] = S_j 规定了栈顶状态为 S_i 且遇到输入符号 a 时应执行的动作。状态转换表 GOTO[S_i, X] = S_j 表示当状态栈顶为 S_i 且文法符号栈顶为 X 时应转向状态 S_j 。

LR 分析器的工作过程以格局的变化来反映。格局的形式为：(栈，剩余输入，动作)。分析是从某个初始格局开始的，经过一系列的格局变化，最终达到接受格局，表明分析成功。或者达到出错格局，表明发现一个语法错误。因此，开始格局的剩余输入应该是全部的输入序列，而接受格局中的剩余输入应该为空，任何其他格局或者出错格局中的剩余输入应该是全部输入序列的一个后缀。

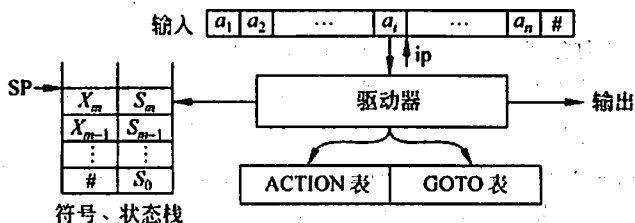


图 4-17 LR 分析器示意图

在 LR 分析过程中，改变格局的动作有以下 4 种。

- 移进 (shift)：当 ACTION[S_i, a] = S_j 时，把 a 移进文法符号栈并转向状态 S_j 。

- 归约(reduce): 当在文法符号栈顶形成句柄 β 时,把 β 归约为相应产生式 $A \rightarrow \beta$ 的非终结符 A 。若 β 的长度为 r (即 $|\beta|=r$),则将文法符号栈顶的 r 个符号弹出,然后将 A 压入文法符号栈中。
- 接受(accept): 当文法符号栈中只剩下文法的开始符号 S ,并且输入符号串已经结束(当前输入符是“#”),分析成功。
- 报错(error): 当输入串中出现不该有的文法符号时,就报错。

LR 分析器的核心部分是分析表的构造,这里不再详述。

5. 语法制导翻译和中间代码生成

程序语言的语义分为静态语义和动态语义。描述程序语义的形式化方法主要有属性文法、公理语义、操作语义和指称语义等,其中属性文法是对上下文无关文法的扩充。目前应用最广泛的静态语义分析方法是语法制导翻译,其基本思想是将语言结构的语义以属性的方式赋予代表此结构的文法符号,而属性的计算以语义规则的形式赋予文法的产生式。在语法分析的推导或归约步骤中,通过语义规则实现对属性的计算,以达到对语义的处理。本节主要讨论以属性为基础的程序静态语义分析和中间代码的生成。

1) 中间代码

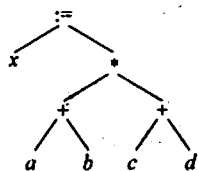
从原理上讲,源程序在进行了语义分析之后就可以直接生成目标代码,但由于源程序与目标代码的逻辑结构往往差别很大,特别是考虑到具体机器指令系统的特点,要使翻译一次到位很困难。另外,用语法制导方式机械生成的目标代码往往是烦琐和低效的,因此有必要设计一种中间代码,将源程序首先翻译成中间代码形式,以利于进行与机器无关的优化处理。由于中间代码实际上也起着编译器前端和后端分水岭的作用,使用中间代码后也有助于提高编译程序的可移植性。常用的中间代码有后缀式、三元式、四元式和树形等形式。

(1) 后缀式(逆波兰式)。

逆波兰式是波兰逻辑学家卢卡西维奇(Lukasiewicz)发明的一种表示表达式的方法。他把运算符写在运算对象的后面,例如,把 $a+b$ 写成 $ab+$,所以也称为后缀式。这种表示法的优点是根据运算对象和算符的出现次序进行计算,不需要使用括号。用栈结构实现后缀式的计算是很方便的。一般的方法是:自左至右扫描后缀式,遇到运算对象时就将其压入栈中,遇到 k 元运算符时就从栈中弹出 k 项进行运算,并将结果压入栈中。当表达式被扫描完时,栈顶元素就是表达式的运算结果。对于表达式 $x := (a+b) * (c+d)$,其后缀式为“ $xab+cd+*:=$ ”。

(2) 树形表示。

我们也可以用树形数据结构来表示一个表达式或语句。例如表达式 $x := (a+b) * (c+d)$ 的树形表示为:



(3) 三元式表示。

三元式是由运算符 OP、第一运算对象 ARG1 和第二运算对象 ARG2 组成的。三元式出现的先后顺序和表达式各部分的计算顺序是一致的。表达式 $x := (a + b) * (c + d)$ 的三元式表示为：

- ① (+, a, b)
- ② (+, c, d)
- ③ (*, ①, ②)
- ④ (: =, ③, x)

(4) 四元式表示。

四元式是一种比较普遍采用的中间代码形式，其组成成分为运算符 OP、第一运算对象 ARG1、第二运算对象 ARG2 和运算结果 RESULT。其中，运算对象和运算结果有时指用户自定义的变量，有时指编译程序引入的临时变量，RESULT 总是一个新引进的临时变量，用来存放运算结果。表达式 $x := (a + b) * (c + d)$ 的四元式表示为：

- ① (+, a, b, t1)
- ② (+, c, d, t2)
- ③ (*, t1, t2, t3)
- ④ (: =, t3, _, x)

2) 常见语法单位的翻译

常见的语法单位主要有：算术表达式、布尔表达式、赋值语句、控制语句（分支和循环）等。对于不同的结构，有不同的处理方法，但翻译程序的构造原理是相似的。

对于各种语法单位的语法制导翻译，一般是在相应的语法规则中加入适当的语义处理规则。下面先说明翻译过程中要使用的语义变量和语义过程。

- Entry(id)的功能是：在符号表中查找标识符 id 以获取它在表中的位置（入口）。
- S.code：语句 S 被翻译后形成的代码序列。
- E.place：与非终结符 E 相联系的语义变量，表示存放 E 值的变量名在符号表的入口或整数码（若此变量是临时变量）。
- E.tc：当表达式 E 的值为真时控制流转向的语句标号（四元式的地址编号）。
- E.fc：当表达式 E 的值为假时控制流转向的语句标号（四元式的地址编号）。

- GEN(OP, ARG1, ARG2, RESULT)的功能是：把四元式(OP, ARG1, ARG2, RESULT)填进四元式表中。
- NXQ：表示下一个将要形成但尚未形成的四元式的地址(编号)。NXQ的初值为1,每执行一次GEN, NXQ就自动累增1。
- Merg(P1, P2)的功能是：把以P1和P2为链首的两条链合并为一条链,并返回合并后的链首指针。
- Backpatch(p, t)的功能是：把p所链接的每条四元式的第四项都以t作为值进行填充。
- Newtemp：生成一个新的临时存储单元。

拉链与回填的基本思想是当一个四元式中存在尚不确定的转向地址时,将所有转向同一地址的四元式连接成一个链表。一旦转向地址被确定,则沿此链向所有的四元式回填该地址。

下面简单说明程序语言中常见结构的语法制导翻译方法。

(1) 赋值语句及简单算术表达式的翻译。

下面的产生式描述了由简单变量、算术加、乘运算、取负运算和圆括号构成的简单算术表达式,以及将一个算术表达式赋值给一个简单变量的赋值语句的构成形式。

$A \rightarrow id := E$

$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid id$

为该文法编写语义子程序如下,其中,若某条产生式中相同的文法符号多次出现,则加一个上标序号以示区别,@表示一元取负运算符。

$A \rightarrow id := E \quad \{GEN(:=, E.place, _, Entry(id))\}$

$E \rightarrow E^{(1)} + E^{(2)} \quad \{E.place := Newtemp; GEN(+, E^{(1)}.place, E^{(2)}.place, E.place)\}$

$E \rightarrow E^{(1)} * E^{(2)} \quad \{E.place := Newtemp; GEN(*, E^{(1)}.place, E^{(2)}.place, E.place)\}$

$E \rightarrow -E^{(1)} \quad \{E.place := Newtemp; GEN(@, E^{(1)}.place, _, E.place)\}$

$E \rightarrow (E^{(1)}) \quad \{E.place := E^{(1)}.place\}$

$E \rightarrow id \quad \{E.place := Entry(id)\}$

(2) 布尔表达式的翻译。

布尔表达式常用于表示控制结构中的条件,可采用直接计算和短路计算两种方式进行计算。直接计算是指布尔表达式中的每个因子都进行运算,而短路计算指只要表达式的值能够确定下来,就停止计算,并非表达式中所有的运算对象都进行运算。下面以短路计算方式为例介绍布尔表达式作为控制条件时的翻译方法。

布尔表达式的形式定义(文法)为：

$E \rightarrow E \text{ and } E \mid E \text{ or } E \mid \text{not } E \mid (E) \mid \text{id} \mid \text{id rel op id}$

其中, rel op 代表关系运算符(<、>、≤、≥、=、≠),运算符的优先级和结合律遵照通常的习惯。

$E \rightarrow E^{(1)}$ and $E^{(2)}$, $E \rightarrow E^{(1)}$ or $E^{(2)}$ 的代码结构分别如图 4-18 和图 4-19 所示。

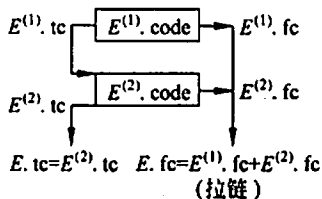


图 4-18 $E \rightarrow E^{(1)}$ and $E^{(2)}$ 的代码结构

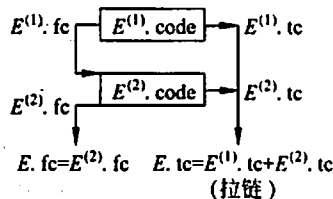


图 4-19 $E \rightarrow E^{(1)}$ or $E^{(2)}$ 的代码结构

对文法进行改写并编写语义子程序如下所示。

对于 $E \rightarrow E^{(1)}$ and $E^{(2)}$, 为记住 $E^{(2)}$ 的第一条四元式的地址, 需改写为:

```

 $E^A \rightarrow E^{(1)}$  and      { Backpatch( $E^{(1)}$ . tc, NXQ);           //回填真出口
                         $E^A$ . fc :=  $E^{(1)}$ . fc;           //传假出口
                        }
 $E \rightarrow E^A E^{(2)}$       {  $E$ . tc :=  $E^{(2)}$ . tc;           //传真出口
                         $E$ . fc := Merg( $E^A$ . fc,  $E^{(2)}$ . fc) //合并假出口
                        }

```

对于 $E \rightarrow E^{(1)}$ or $E^{(2)}$, 为记住 $E^{(2)}$ 的第一条四元式的地址, 须改写为:

```

 $E^V \rightarrow E^{(1)}$  or      { Backpatch( $E^{(1)}$ . fc, NXQ);           //回填假出口
                         $E^V$ . tc :=  $E^{(1)}$ . tc;           //传真出口
                        }
 $E \rightarrow E^V E^{(2)}$       {  $E$ . tc := Merg( $E^V$ . tc,  $E^{(2)}$ . tc); //合并真出口
                         $E$ . fc :=  $E^{(2)}$ . fc           //传假出口
                        }
 $E \rightarrow \text{not } E^{(1)}$     {  $E$ . tc :=  $E^{(1)}$ . fc;  $E$ . fc :=  $E^{(1)}$ . tc } //交换真、假出口
 $E \rightarrow (E^{(1)})$         {  $E$ . tc :=  $E^{(1)}$ . tc;  $E$ . fc :=  $E^{(1)}$ . fc } //传真、假出口
 $E \rightarrow \text{id}$            {  $E$ . tc := NXQ;  $E$ . fc := NXQ+1;
                        GEN(jnz, Entry(id), _, 0); GEN(j, _, _, 0); }
 $E \rightarrow \text{id}^{(1)} \text{ rel op id}^{(2)}$  {  $E$ . tc := NXQ;  $E$ . fc := NXQ+1;
                        GEN(jrel op, Entry( $\text{id}^{(1)}$ ), Entry( $\text{id}^{(2)}$ ), 0);
                        GEN(j, _, _, 0); }

```

(3) 常见语句的翻译。

这里只列举条件语句、while 循环语句和赋值语句的翻译方法, 其形式定义为:

$S \rightarrow A \mid \text{if } E \text{ then } S^{(1)} \mid \text{if } E \text{ then } S^{(1)} \text{ else } S^{(2)} \mid \text{while } E \text{ do } S^{(1)}$

用语义变量 $S.\text{chain}$ 记录语句结束后的转向地址, $W.\text{quad}$ 用来记住 while 语句所对应的第一条四元式的地址。回填工作将在处理 S 的外层环境的某个时刻完成。条件语句和循环语句的代码结构如图 4-20 所示。

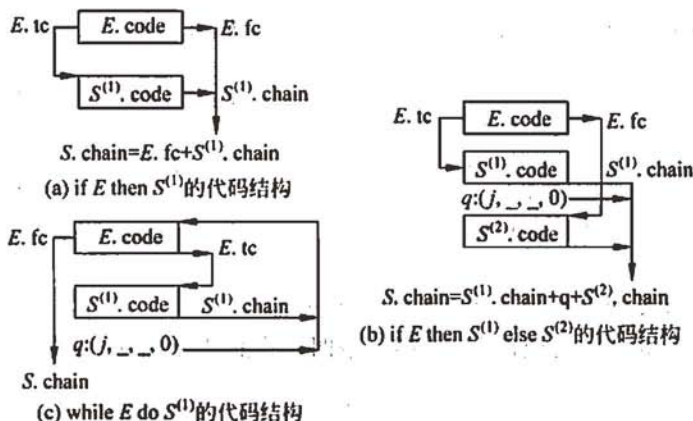


图 4-20 控制语句的代码结构

```

C → if E then { Backpatch(E.tc, NXQ);           // 填真出口
                C.chain := E.fc;                 // 传假出口 }
S → CS(1)      { S.chain := Merg(C.chain, S(1).chain); } // 拉链
Tp → CS(1) else { q := NXQ; GEN(j, -, -, 0);
                Tp.chain := Merg(S(1).chain, q);    // 拉链
                Backpatch(C.chain, NXQ);          // 填假出口
            }
S → TpS(2)    { S.chain := Merg(Tp.chain, S(2).chain); } // 拉链
W → while    { W.quad := NXQ; }                     // 记录 while 语句的首地址
Wd → W E do { Backpatch(E.tc, NXQ);                 // 填真出口
                Wd.chain := E.fc;                   // 传假出口
                Wd.quad := W.quad;                  // 传首地址
            }
S → WdS(1)    { Backpatch(S(1).chain, Wd.quad);      // S(1) 执行完后转向 E
                GEN(j, -, -, Wd.quad);
                S.chain := Wd.chain;                 // S 的出口
            }
S → A        { S.chain := 0; } // 赋值语句不含其他非常规出口(无出口链),
    
```

3) 动态存储分配

过程(函数)说明和过程(函数)调用是程序中一种常见的语法结构,绝大多数语言都

含有这方面的内容。过程说明和调用语句的翻译,有赖于形式参数与实际参数结合的方式以及数据空间的分配方式。

由于各种语言的不同特点,在目标程序运行时,对存储空间的分配和组织有不同的要求,在编译阶段应产生相应的目标程序来满足不同的要求。需要分配存储空间的对象有基本数据类型(如整型、实型和布尔型等)、结构化数据类型(如数组和记录等)和连接数据(如返回地址和参数等)等。分配的依据是名字的作用域和生存期的定义规则,分配的策略有静态和动态两大类。

如果编译时就能确定目标程序运行时所需的全部数据空间的大小,则在编译时就安排好目标程序运行时的全部数据空间,并确定每个数据对象的存储位置。这种分配策略称为静态存储分配。FORTRAN 语言可以完全采用静态存储分配策略。

如果一个程序语言允许递归过程和可变数据结构,那么,就需采用动态存储分配技术。动态存储分配策略的实现有两种方式:栈分配方式和堆分配方式。栈式动态存储分配策略是将程序的数据空间设计为一个栈,存储空间遵循先分配后释放的原则进行管理,适用于 Pascal, C, ALGOL 之类的语言。每当调用一个过程时,它所需的数据空间就在栈顶分配。每当过程工作结束时,就释放这部分空间。如果一个程序语言为用户提供自由申请和退还数据空间的机制(如 C++ 中的 new/delete, Pascal 中的 new),或者不仅有过程而且有进程的程序结构,即空间的使用未必服从“先申请后释放”的原则,那么栈式动态存储分配方案就不适用了,这种情况下可使用堆分配技术。

6. 中间代码优化

优化就是对程序进行等价变换,使变换后的程序能生成更有效的目标代码。所谓等价,是指不改变程序的运行结果。而有效是指目标代码运行时间较短,占用的存储空间较少。优化可在编译的各个阶段进行。最主要的做法是在目标代码生成以前,对中间代码进行优化,这类优化不依赖于具体的计算机。由于对目标代码的优化涉及具体的计算机系统,所以在此不作讨论。

为了叙述方便,下面我们将四元式改写为更直观表示形式:

- 将 (op, B, C, A) 写成 $A := B \text{ op } C$;
- 将 $(jrel, B, C, L)$ 写成 $\text{if } B \text{ rel op } C \text{ goto } L$;
- 将 $(j, _, _, L)$ 写成 $\text{goto } L$ 。

根据优化所涉及的程序范围,可分为局部优化、循环优化和全局优化。下面简单介绍局部优化和循环优化的主要技术。

1) 局部优化

局部优化是在基本块上进行的优化。所谓基本块,是指程序中一个顺序执行的语句(四元式)序列,其中只有一个入口和一个出口。入口就是其中的第一条语句,出口是最后

一条语句。对一个给定的程序,可以将其划分成一系列的基本块,在各基本块范围内,分别进行优化。

(1) 基本块的划分。

基本块的入口语句是下列 3 种情况之一: 程序的第一条语句; 条件转移或无条件转移语句转移到的目标语句; 紧跟在条件转移语句后面的语句。

求出程序中的各条入口语句后,对于每一条入口语句,其所属的基本块由该入口语句到下一条入口语句(不包括下一入口语句),到一条转移语句(包括该转移语句),或到一个停机语句(包括该停机语句)之间的语句序列组成。凡未能纳入某一基本块的语句,都是程序控制流无法到达的语句,可将它们从程序中删除。

在一个基本块内,通常可实行以下 3 种优化: 合并已知量,删除无用赋值和删除多余运算。

(2) 优化方法。

一个基本块可用以用一个 DAG(有向无环)图表示。

【例 4.5】 下面是一个基本块的代码序列,根据该代码序列构造的 DAG 如图 4-21 所示。

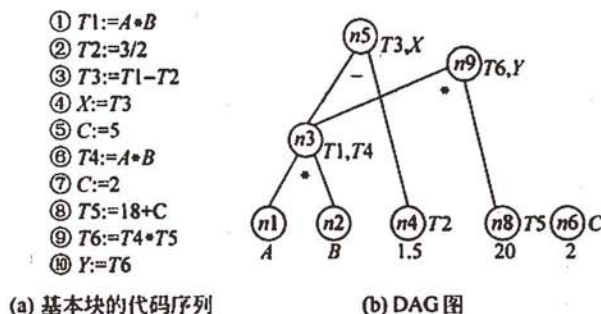


图 4-21 基本块的 DAG 图

根据图 4-21 所示的 DAG 图,依照结点的构造次序,重新写成的四元式序列如下:

- ① $T1 := A * B$
- ② $T2 := 1.5$
- ③ $T3 := T1 - 1.5$
- ④ $X := T3$
- ⑤ $T4 := T1$
- ⑥ $C := 2$
- ⑦ $T5 := 20$

⑧ $T6 := T1 * 20$

⑨ $Y := T6$

在重建基本块时,进行了合并已知量、无用赋值和删除多余运算的处理,使新的基本块效率更高。

2) 控制流分析和循环优化

(1) 控制流图。

为了描述程序中各基本块间的关系,可以用控制流图作为工具。一个程序的控制流图是一个有向图,其结点是程序中的基本块,它有惟一的首结点,即包含程序第一条语句的基本块。从首结点出发,控制流图中的每个结点都存在路径。

由程序的各基本块构造相应控制流图的方法是:对于程序中的两个基本块 B_i 和 B_j ,若 B_j 紧接着 B_i 被执行,则从 B_i 引一条有向边到 B_j ,称 B_i 是 B_j 直接前驱、 B_j 是 B_i 的直接后继。确定 B_i 是 B_j 直接前驱的规则是:

- 按程序中基本块的自然排列次序, B_i 紧跟在 B_j 之后,且 B_i 的出口语句不是无条件转移或停语句。
- B_i 的出口语句是条件转移或无条件转移语句,且转向的目标就是 B_j 的入口语句。

【例 4.6】 对下面的程序段构造的控制流图如图 4-22 所示。

- ① $i := 1$
- ② $j := 10$
- ③ read k
- ④ L: $x := k * i$
- ⑤ $y := j * i$
- ⑥ $z := x * y$
- ⑦ write j
- ⑧ $i := i + 1$
- ⑨ if $i < 100$ goto L
- ⑩ halt

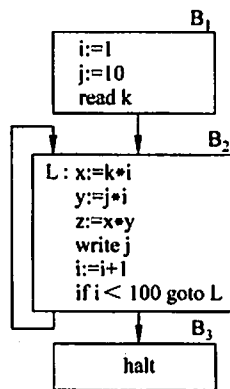


图 4-22 控制流图

(2) 循环优化。

通常,可以把循环理解为程序中的一个能被重复执行的代码序列。我们把控制流图中具有以下性质的结点序列称为一个循环:

- 这个结点序列有惟一的入口结点,使得从循环外到达循环内的任何结点都要通过此结点。
- 这个结点序列是强连通的,也就是说其中任意两个结点之间必有一条路径,且路径上的结点都属于该结点序列。如果序列中只有一个结点,则必有一条有向边从

该结点引到自身。

因此,循环就是控制流图中具有惟一入口结点的强连通子图。从循环外进入循环时,必须首先经过循环的入口结点。

基于循环的优化处理有:代码外提、削弱运算强度、删除归纳变量等。

- 代码外提。

循环中的代码要随着循环反复地执行,但其中某些运算的结果并不因循环而改变,对于这些运算,我们可以将其提取到循环之外。这样,程序的运行结果仍保持不变,而程序的执行效率却提高了。这种优化处理称为代码外提。

在进行代码外提时,我们在循环入口结点之前建立一个新结点(前置结点),它以循环入口结点为惟一后继,原来从循环外引到循环入口结点的有向边改成引到循环的前置结点。因为循环的入口结点是惟一的,所以一个循环的前置结点也是惟一的。

- 强度削弱。

强度削弱是指把程序中执行时间较长的运算替换为执行时间较短的运算。削弱强度不仅可针对乘法运算(将循环中的乘法运算用加法运算来替换),也可针对加法运算。

如果循环中有 I 的递归赋值 $I := I \pm C$ (C 为循环不变量),且循环中 T 的赋值运算可转化为 $T := K * T \pm C_1$ (K 和 C_1 为循环不变量),那么可对 T 的赋值运算进行强度削弱优化处理。

- 删除归纳变量。

如果循环中变量 I 只有惟一的形如 $I := I \pm C$ 的赋值,且其中 C 为循环不变量,则称 I 为循环中的基本归纳变量。

如果 I 是循环中惟一的基本归纳变量, J 在循环中的定值总是可以转化为 I 的同一线性函数,也就是 $J := C_1 * I \pm C_2$,其中 C_1 和 C_2 都是循环不变量,则称 J 是归纳变量,并称它与 I 同族。一个基本归纳变量也是归纳变量。删除归纳变量一般在强度削弱之后进行。

下面对图 4-22 所作的优化处理进行说明。对于循环中的代码“ $x := k * i$ ”和“ $y := j * i$ ”,由于 j 和 k 在循环中不改变值,所以对 x 和 y 可以进行强度削弱处理,优化后程序的控制流图如图 4-23(a)所示。

由于 i 是循环中的基本归纳变量, x 、 y 是与 i 同族的归纳变量,且有线性关系“ $x := k * i$ ”和“ $y := j * i$ ”,所以,“ $i < 100$ ”完全可以用“ $x < 100 * k$ ”或“ $y < 100 * j$ ”代替。于是,进一步优化的结果如图 4-23(b)所示。

将循环中的不变运算“ $t1 := 100 * k$ ”提取到循环的前置结点中,程序的流图如图 4-23(c)所示。

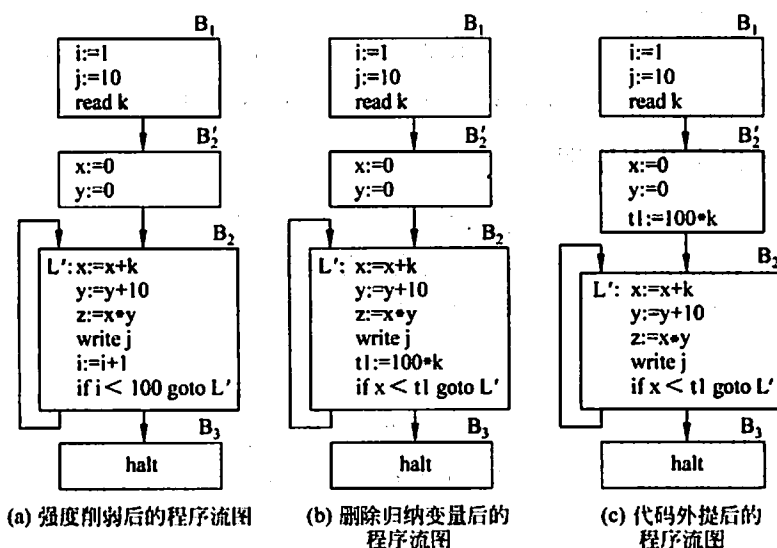


图 4-23 优化结果示意图

7. 目标代码生成

目标代码的生成由代码生成器实现。代码生成器以经过语义分析或优化后的中间代码为输入,以特定的机器语言或汇编代码为输出。代码生成所需考虑的主要问题有以下几个。

1) 中间代码形式

中间代码有多种形式,其中树与后缀表示形式适用于解释器,而编译器多采用与机器指令格式较接近的四元式形式。

2) 目标代码形式

目标代码可以分为两大类:汇编语言形式和机器指令形式。机器指令形式的目标代码根据需求的不同又可以分为绝对机器指令代码和可再定位机器代码。绝对机器代码的优点是可以立即执行,一般应用于一类称为 load-and-go 形式的编译模式,即编译后立即执行,不形成存在外存上的目标代码文件。可再定位机器代码的优点是目标代码可以被任意链接并装入内存的任意位置,是编译器采用较多的代码形式。汇编语言作为一种中间输出形式,便于进行分析和测试。

3) 寄存器的分配

由于存取寄存器的速度远快于存取内存单元的速度,所以总是希望尽可能多地使用寄存器存储数据。而寄存器的个数是有限的,因此,如何分配及使用寄存器,是目标代码生成时需要着重考虑的问题。

4) 计算次序的选择

代码执行的效率会随计算次序的不同有较大的差别。在生成正确的目标代码的前提下,适当地安排计算次序并优化代码序列,也是生成目标代码时要考虑的重要因素之一。

4.2.3 解释程序基本原理

解释程序是另一种语言处理程序,在词法、语法和语义分析方面与编译程序的工作原理基本相同,但在运行用户程序时,它直接执行源程序或源程序的内部形式。因此,解释程序不产生源程序的目标程序,这是它和编译程序的主要区别。图 4-24 显示了以解释方式实现高级语言的三种方式。

源程序被直接解释执行的处理方式如图 4-24 中的标记 A 所示。这种解释程序对源程序逐个字符进行检查,然后执行程序语句规定的动作。例如,如果扫描到字符串序列:

GOTO L

解释程序就开始搜索源程序中标号 L 后面紧跟冒号“:”的出现位置。这类解释程序通过反复扫描源程序来实现程序的运行,运行效率很低。

解释程序也可以先将源程序翻译成某种中间代码形式,然后对中间代码进行解释,实现用户程序的运行。这种翻译方式如图 4-24 中的标记 B 和 C 所示。通常,在中间代码和高级语言的语句间存在一一对应的关系。APL 和 SNOBOL4 的很多实现就采用这种方法。解释方式 B 和 C 的不同之处在于中间代码的级别。在方式 C 下,解释程序采用的中间代码更接近于机器语言。在这种实现方案中,高级语言和低级中间代码间存在着“1:n”的对应关系。Pascal-P 解释系统是这类解释程序的一个实例。它在词法分析、语法分析和语义分析的基础上,先将源程序翻译成 P-代码,再由一个非常简单的解释程序来解释执行这种 P-代码。这类系统具有比较好的可移植性。

下面简要描述一下解释程序的结构。这类系统通常可以分成两部分。第一部分是分析部分,包括通常的词法分析、语法分析和语义分析程序。经语义分析后把源程序翻译成中间代码,中间代码常采用逆波兰表示形式。第二部分是解释部分,用来对第一部分产生的中间代码进行解释执行。下面简要介绍第二部分的工作原理。

设用数组 MEM 模拟计算机的内存,源程序的中间代码和解释部分的各个子程序都存放在数组 MEM 中。全局变量 PC 是一个程序计数器,它记录了当前正在执行的中间代码的位置。这种解释程序的结构可以由下面两部分组成:

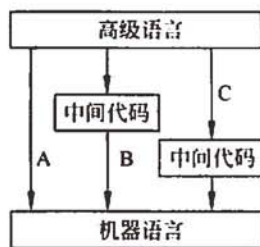


图 4-24 解释器类型示意图

(1) $PC := PC + 1$;

(2) 执行位于 $opcode-table[MEM[PC]]$ 的子程序(解释子程序执行后返回到前面)。

用一个简单例子来说明其工作原理。设两个实型变量 A 和 B 进行相加的中间代码是:

```
start:  Ipush
        A
        Ipush
        B
        Iaddreal
```

其中,中间代码 Ipush 和 Iaddreal 实际上都是 opcode-table 表的索引值(即位移),而该表的单元中存放着对应的解释子程序的起始地址,A 和 B 都是 MEM 中的索引值。解释部分开始执行时,PC 的值为 start-1。

```
opcode-table[Ipush] = push
opcode-table[Iaddreal] = addreal
```

解释部分可表示如下:

```
interpreter-loop:  PC := PC + 1;
                  goto opcode-table[MEM[PC]];
push:  PC := PC + 1;
      stackreal(MEM[MEM[PC]]);
      goto interpreter-loop;
addreal:  stackreal(popreal() + popreal());
      goto interpreter-loop;
... (其余为各解释子程序)
```

其中,stackreal()表示把相应值压入栈中,而 popreal()表示取得栈顶元素值并弹出栈顶元素。

对于高级语言的编译和解释工作方式,可从以下几个方面进行比较。

1) 效率

编译比解释方式可能取得更高的效率。

一般情况下,在解释方式下运行程序时,解释程序可能需要反复扫描源程序。例如,每一次引用变量都要进行类型检查,甚至需要重新进行存储分配,从而降低了程序的运行速度。在空间上,以解释方式运行程序需要更多的内存,因为系统不但需要为用户程序分配运行空间,而且要为解释程序及其支撑系统分配空间。

在编译方式下,编译程序除了对源程序进行语法和语义分析外,还要生成源程序的目标代码并进行优化,所以这个过程比解释方式需要更多的时间。虽然与仔细写出的机器程序相比,一般由编译程序创建的目标程序运行的时间更长,需要占用的存储空间更多。但源程序只需要被编译程序翻译一次,就可以多次运行。因此总体来讲,编译方式比解释方式可能取得更高的效率。

2) 灵活性

由于解释程序需要反复检查源程序,这也使得解释方式能够比编译方式更灵活。当解释器直接运行源程序时,“在运行中”修改程序就成为可能。例如,增加语句或者修改错误等。另外,当解释器直接在源程序上工作时,它可以对错误进行更精确地定位。

3) 可移植性

解释器一般也是用某种程序设计语言编写的,因此只要对解释器重新进行编译,就可以使解释器运行在不同的环境中。

由于编译和解释的方法各有特点,因此现有的一些编译系统既提供编译方式,也提供解释方式,甚至将两种方式结合在一起。例如,在 Java 虚拟机上发展的一种“compiling-just-in-time”技术,就是在代码第一次运行时进行编译,其后的运行就不再进行编译了。

第5章 网络基础知识

计算机网络是由多台计算机组成的系统,与传统的单机系统、多机系统相比有很大的区别。计算机网络的结构、功能、组成以及实现技术更复杂,维护起来难度更大。本章将简要介绍计算机网络的体系结构、网络应用、网络构建与网络安全方面的基本内容,涉及到网络有关的硬件和应用知识。

5.1 网络概述

由于对信息共享和信息传递应用的迫切需求,计算机应用日益渗透到各行各业。计算机已不仅仅是一种计算的工具,而是更多地应用于信息的收集、处理和传输。计算机参与社会信息化的进程,促进了信息产业的发展。而应用需求又推动了计算机技术的不断更新,越来越多的领域急需计算机在一定范围内“联合起来”,进行群体工作。在此环境下计算机网络应运而生了。计算机网络是计算机技术与通信技术相结合的产物,它实现了远程通信、远程信息处理和资源共享。经过几十年的发展,计算机网络已由早期的“终端-计算机网”、“计算机-计算机网”成为现代具有统一网络体系结构的计算机网络。

5.1.1 计算机网络的概念

计算机网络是计算机技术与通信技术日益发展和密切结合的产物,它的发展过程大致可以划分为以下4个阶段。

1) 具有通信功能的单机系统

该系统又称终端-计算机网络,是早期计算机网络的主要形式。它将一台计算机经通信线路与若干终端直接相连。美国于20世纪50年代建立的半自动地面防空系统SAGE就属于这一类网络。它把远距离的雷达和其他测量控制设备的信息通过通信线路送到一台旋风型计算机上进行处理和控制在,首次实现了计算机技术与通信技术的结合。

2) 具有通信功能的多机系统

对终端-计算机网进行改进:在主计算机的外围增加一台计算机,专门用于处理终端的通信信息及控制通信线路,并能对用户的作业进行某些预处理操作,这台计算机称为“前端处理机”或“通信控制处理机”。在终端设备较集中的地方设置一台集中器,终端通过低速线路先汇集到集中器上,然后再用高速线路将集中器连到主机上,这就形成多机

系统。

3) 以共享资源为目的的计算机网络

具有通信功能的多机系统是计算机-计算机网络,它是由若干台计算机互联而构成的系统,即利用通信线路将多台计算机连接起来,在计算机之间进行通信。该网络有两种结构形式:一种形式是主计算机通过通信线路直接互联,其中主计算机同时承担数据处理和通信工作;另一种形式是通过通信控制处理机间接地把各主计算机连接起来,其中通信处理机和主计算机分工,前者负责网络上各主计算机间的通信处理和控制在,后者是网络资源的拥有者,负责数据处理,它们共同组成资源共享的计算机网络。20 世纪 70 年代,美国国防部高级研究计划局所研制的 ARPAnet 是计算机-计算机网络的典型代表。最初该网仅由 4 台计算机连接而成,到 1975 年,已连接 100 多台不同型号的大型计算机。ARPAnet 成为第一个完善地实现分布式资源共享的网络,为计算机网络的发展奠定了基础。

在这期间,国际标准化组织(ISO)提出了开放系统互连参考模型(open system interconnection model, OSI/RM)。该模型定义了异种机联网所应遵循的框架结构。OSI/RM 很快得到了国际上的认可,并为许多厂商所接受。由此使计算机网络的发展进入了新的阶段。

4) 以局域网及互联网为支撑环境的分布式计算机系统

局域网是继远程网之后发展起来的,它继承了远程网的分组交换技术和计算机的 I/O 总线结构技术。局域网的发展也促使计算机网络的模式发生了变革,即由早期的以大型机为中心的集中式模式转变为由微机构成的分布式计算机模式。

计算机网络的定义随网络技术的更新可从不同的角度给以描述。目前人们已公认的有关计算机网络的定义是:利用通信设备和线路将地理位置分散的、功能独立的自主计算机系统或由计算机控制的外部设备连接起来,在网络操作系统的控制下,按照约定的通信协议进行信息交换,实现资源共享的系统。

定义中涉及的“资源”应该包括硬件资源(CPU、大容量的磁盘、光盘以及打印机等)和软件资源(语言编译器、文本编辑器、各种软件工具 and 应用程序等)。

计算机网络提供的主要功能有以下 4 个方面。

(1) 数据通信,通信或数据传输是计算机网络的主要功能之一,用以在计算机系统之间传送各种信息。利用该功能,地理位置分散的生产单位和业务部门可通过计算机网络连接在一起进行集中控制和管理。也可以通过计算机网络传送电子邮件、发布新闻消息和进行电子数据交换,极大地方便了用户,提高了工作效率。

(2) 资源共享,资源共享是计算机网络最有吸引力的功能。通过资源共享,可使网络中分散在异地的各种资源互通有无,分工协作,从而大大提高系统资源的利用率。资源共

享包括软件资源共享和硬件资源共享。

- 硬件资源共享：可以在全网范围内提供对处理机、存储器和输入输出设备等资源的共享,特别是对一些较高级和昂贵设备的共享,如巨型计算机、高分辨率打印机和大型绘图仪等,从而使用户节省投资,也便于资源和任务的集中管理以及分担负荷。
- 软件资源共享：在局域网上允许用户共享文件服务器上的程序和数据;在互联网上允许用户远程访问各种类型的数据库,可以得到网络文件传送服务、远程管理服务和远程文件访问。从而可以避免软件研制上的重复劳动以及数据资源的重复存储,且便于集中管理。

(3) 负载均衡,在计算机网络中可进行数据的集中处理或分布式处理,一方面可以通过计算机网络将不同地点的主机或外设采集到的数据信息送往一台指定的计算机,在此计算机上对数据进行集中和综合处理,通过网络在各计算机之间传送原始数据和计算结果;另一方面当网络中某台计算机任务过重时,可将任务分派给其他空闲的多台计算机,使多台计算机相互协作,均衡负载,共同完成任务。当今计算方式的一种新趋势——协同式计算,就是利用网络环境的多台计算机来共同处理一个任务。客户机/服务器模式也是实现这一功能的一种应用。

(4) 高可靠性,指在一个系统中组成网络的各台计算机可以彼此互为后备机,一旦某台计算机出现故障,故障机的任务就可由其他计算机代为处理,从而提高系统的可靠性。避免了单机无后备使用的情况下,计算机出现故障而导致系统瘫痪的现象,从而大大提高了系统的可靠性。

借助于计算机网络,在各种功能软件的支持下,人类可以进行高速的异地电子信息交换,并获得多种服务,如新闻浏览和信息检索、传送电子邮件、多媒体电信服务、远程教育、网上营销、网上娱乐和远程医疗诊断等。

按照数据通信和数据处理的功能,计算机网络可分为内层通信子网和外层资源子网两层。如图 5-1 所示。通信子网(图中虚线内)的节点计算机和高速通信线路组成独立的数据系统,承担全网的数据传输、交换、加工和变换等通信处理工作,即将一个计算机的输出信息传送给另一个计算机。资源子网(图中点划线内虚线外)包括计算机、终端、通信子网接口设备、外部设备(如打印机、磁带机和绘图机等)及各种软件资源等,它负责全网的数据处理并向网络用户提供网络资源及网络服务。

通信子网和资源子网的划分,完全符合国际标准化组织(ISO)所制定的开放式系统互连参考模型 OSI 的思想。其中通信子网对应于 OSI 中的低 3 层(物理层、数据链路层和网络层),而资源子网对应于 OSI 中的高 3 层(会话层、表示层和应用层)。这种划分将通信子网的任务从主机中抽取出来,由通信子网中的设备专门解决数据传输和通信控制

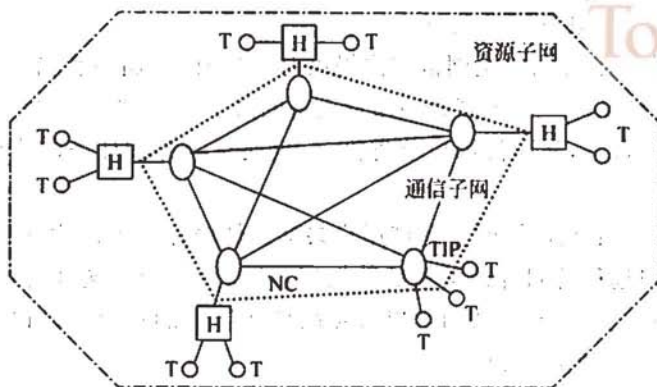


图 5-1 通信子网和资源子网关系图

H——主计算机 T——终端 TIP——集线器 NC——通信处理

问题。而资源子网中的计算机可集中精力处理数据,从而提高主机效率和网络的整体性能。

5.1.2 计算机网络的分类

计算机网络的分类方式很多,按照不同的分类原则,可以得到各种不同类型的计算机网络。例如,按通信距离可分为广域网(WAN)、局域网(LAN)和城域网(MAN);按信息交换方式可分为电路交换网、分组交换网和综合交换网;按网络拓扑结构可分为星形网、树形网、环形网和总线网;按通信介质可分为双绞线网、同轴电缆网、光纤网和卫星网等;按传输带宽可分为基带网和宽带网;按使用范围可分为公用网和专用网;按速率可分为高速网、中速网和低速网;按通信传播方式可分为广播式和点到点式。

这里主要介绍根据计算机网络的覆盖范围和通信终端间的距离分类的局域网、城域网和广域网 3 种情况。各类网络的特征参数如表 5-1 所示。

表 5-1 各类网络的特征参数

网络分类	缩写	分布距离	计算机分布范围	传输速率范围
局域网	LAN	10m 左右	房 间	4Mb/s~1Gb/s
		100m 左右	楼 寓	
		1000m 左右	校 园	
城域网	MAN	10km	城 市	50Kb/s~100Mb/s
广域网	WAN	100km 以上	国家或全球	9.6Kb/s~45Mb/s

1. 局域网

局域网(local area network)是指传输距离有限,传输速度较高,以共享网络资源为目的的网络系统。由于局域网投资规模较小,网络实现简单,故新技术易于推广。与广域网相比,局域网技术发展迅速。局域网的特点如下:

(1) 分布范围有限。加入局域网中的计算机通常处在几千米的距离之内,分布在一个学校或一个企业单位,为本单位使用。入网的计算机及其网络设备局限在一个房间、一栋楼宇或一个园区或校园网之内。一般称为“园区网”或“校园网”。

(2) 有较高的通信带宽,数据传输率高。一般为 1Mb/s 以上,最高已达 1000Mb/s。

(3) 数据传输可靠,误码率低。误码率一般为 $10^{-4} \sim 10^{-6}$ 。

(4) 通常采用同轴电缆或双绞线作为传输介质。跨楼宇时使用光纤。

(5) 拓扑结构简单简洁,大多采用总线、星形和环形等,系统容易配置和管理。网上的计算机一般采用多路控制访问技术或令牌技术访问信道。

(6) 网络的控制一般趋向于分布式,从而减少了对某个节点的依赖性,避免并减小了一个节点故障对整个网络的影响。

(7) 网络通常归单一组织所拥有和使用。不受任何公共网络管理机构的规定约束,容易进行设备的更新和新技术的引用,以不断增强网络功能。

使用局域网技术构建的园区网是连接一个或相距不远的多个建筑物间的多个工作组的计算机网络。最典型的园区网是连接大学各系的校园网。其地理覆盖范围一般在一公里到几公里以内。园区网提供的服务包括:连接各个工作组,访问服务器、高速打印机和绘图仪等昂贵设备,并提供对公用数据库的访问。而企业网用于连接一个公司或企业的所有计算机。它可能覆盖几公里、几十公里甚至几百公里的范围,一般会包括多个局域网。企业网用户可以共享公司其他部门、办公室以及公司总部的信息,并相互传递相关信息或电子邮件,也可以访问中心主机,还可以共享企业网的其他服务。

2. 城域网

城域网(metropolitan area network)是规模介于局域网和广域网之间的一种较大范围的高速网络,一般覆盖临近的多个单位和城市,从而为接入网络的企业、机关、公司及社会单位提供文字、声音和图像的集成服务。城域网规范由 IEEE802.6 协议定义。

3. 广域网

广域网(wide area network)又称远程网。它是指覆盖范围广,传输速率相对较低,以数据通信为主要目的的数据通信网。广域网最根本的特点是:

(1) 分布范围广。加入广域网中的计算机通常处在从数公里到数千公里的地方。因此网络所涉及的范围可为市、地区、省、国家乃至世界。

(2) 数据传输率低。一般为每秒几十兆比特以下。

(3) 数据传输的可靠性随着传输介质的不同而不同。若用光纤, 误码率一般在 $10^{-6} \sim 10^{-11}$ 之间。

(4) 广域网常常借用传统的公共传输网来实现, 因为单独建造一个广域网极其昂贵。

(5) 拓扑结构较为复杂, 大多采用“分布式网络”, 即所有计算机都与交换节点相连, 从而使网络中任何两台计算机都可以进行通信。

广域网的布局不规则, 使得网络的通信控制比较复杂。尤其是使用公共传输网, 要求连接到网上的任何用户都必须严格遵守各种标准和规程。设备的更新和新技术的引用难度较大。广域网可将一个集团公司、团体或一个行业的各处部门和子公司连接起来。这种网络一般要求兼容多种网络系统(异构网络), 包括多种机型、多种网络标准、多种网络连接设备以及多种网络操作系统。

5.1.3 网络的拓扑结构

网络拓扑结构是指网络中通信线路和节点的几何形状, 用以表示整个网络的结构外貌, 反映各节点之间的结构关系。它影响着整个网络的设计、功能、可靠性和通信费用等重要方面, 是计算机网络十分重要的要素。常用的网络拓扑结构有总线、星状、环状、树状和分布式等。

1. 总线结构

总线拓扑结构如图 5-2(a) 所示, 其特点为: 总线拓扑结构中只有一条双向通路, 便于采用广播式传送信息; 总线拓扑结构属于分布式控制, 无须中央处理器, 故结构比较简单; 节点的增删和位置的变动较容易, 变动时不影响网络的正常运行, 系统扩充性能好; 节点的接口通常采用无源线路, 系统可靠性高; 设备少, 价格低, 安装使用方便; 但由于电气信号通路多, 干扰较大, 因此对信号的质量要求高。负载重时, 线路的利用率较低。网上的信息延迟时间不确定, 故障隔离和检测困难。

2. 星状结构

星状结构中, 使用中央交换单元以放射状连接到网中的各个节点, 如图 5-2(b) 所示。中央单元采用电路交换方式在希望通信的两节点间建立专用的路径。通常用双绞线将节点与中央单元进行连接。其特点为: 维护管理容易, 重新配置灵活; 故障隔离和检测容易; 网络延迟时间短; 各节点与中央交换单元直接连通, 各节点之间的通信必须经过中央单元转换; 网络共享能力差; 线路利用率低, 中央单元负荷重。

3. 环状结构

环状结构的信息传输线路构成一个封闭的环状, 各节点通过中继器连入网内, 各中继器间首尾相接, 如图 5-2(c) 所示。信息单向沿环路逐点传送。其特点为: 环形网中信息的流动方向是固定的, 两个节点仅有一条通路, 路径控制简单; 有旁路设备, 节点一旦发生

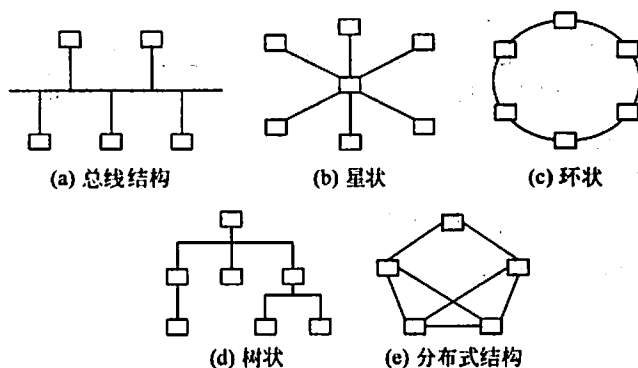


图 5-2 常用的网络拓扑结构

故障,系统自动旁路,可靠性高;信息要串行穿过多个节点,在网中节点过多时传输效率低,系统响应速度慢;由于环路封闭,扩充较难。

4. 树状结构

树状结构是总线形结构的扩充形式,传输介质是不封闭的分支电缆,如图 5-2(d)所示。它主要用于多个网络组成的分级结构中。其特点同总线网。

5. 分布式结构

分布式结构无严格的布点规定和形状,各节点之间有多条线路相连,如图 5-2(e)所示。其特点为:分布式网有较高的可靠性,当一条线路有故障时,不会影响整个系统的工作;资源共享方便,网络响应时间短;由于节点与多个节点连接,故节点的路由选择和流量控制难度大,管理软件复杂;硬件成本高。

广域网与局域网所使用的网络拓扑结构有所不同,广域网多用分布式或树状结构,而局域网常使用总线、环状、星状或树状结构。

5.2 ISO/OSI 网络体系结构

计算机网络是相当复杂的系统,相互通信的两个计算机系统必须高度协调才能正常工作。为了设计这样复杂的计算机网络,人们提出了将网络分层的方法。分层可将庞大而复杂的问题转化为若干较小的局部问题进行处理,从而使问题简单化。

国际标准化组织(International Standard Organization, ISO)在 1977 年成立一个分委员会,专门研究网络通信的体系结构问题,并提出了开放系统互连参考模型,它是一个定义异种计算机连接标准的框架结构。OSI 为连接分布式应用处理的“开放”系统提供了基础。所谓“开放”是指任何两个系统,只要遵守参考模型和有关标准,都能够进行互连。

OSI 采用了层次化结构的构造技术。

ISO 分委员会的任务是定义一组层次和每一层所完成的功能和服务。层次的划分应当从逻辑上将功能分组,层次应该足够多,应使每一层小到易于管理的程度,但也不能太多,否则汇集各层的处理开销太大。

1. ISO/OSI 参考模型

ISO/OSI 的参考模型共有 7 层,如图 5-3 所示。由低层至高层分别为物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

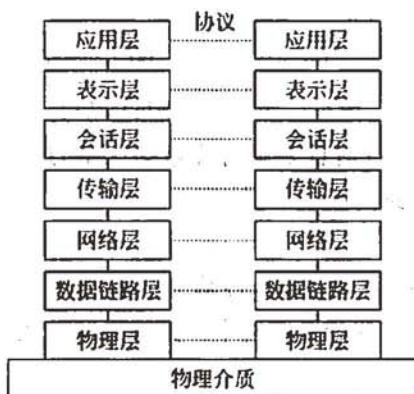


图 5-3 OSI 参考模型

OSI 参考模型具有以下特性:

- 它是一种异构系统互连的分层结构。
- 提供了控制互连系统交互规则的标准框架。
- 定义了一种抽象结构,而并非具体实现的描述。
- 不同系统上的相同层的实体称为同等层实体。
- 同等层实体之间的通信由该层的协议管理。
- 相邻层间的接口定义了原语操作和低层向高层提供的服务。
- 所提供的公共服务是面向连接的或无连接的数据服务。
- 直接的数据传送仅在最低层实现。
- 每层完成所定义的功能,修改本层的功能并不影响其他层。

OSI 参考模型各层的功能简述如下所示。

OSI/RM 中的 1~3 层主要负责通信功能,一般称为通信子网层。上 3 层即 5~7 层属于资源子网的功能范畴,称为资源子网层。传输层起着衔接上下 3 层的作用。具体来说:

(1) 物理层(physical layer): 提供为建立、维护和拆除物理链路所需的机械、电气、功能和规程的特性;提供有关在传输介质上传输非结构的位流及物理链路故障检测指示。

用户要传递信息就要利用一些物理媒体,如双绞线、同轴电缆等,但具体的物理媒体并不在 OSI 的 7 层之内,有人把物理媒体当作第 0 层。物理层的任务就是为它的上一层提供一个物理连接,规定它们的机械、电气、功能和过程特性。如规定使用电缆和接头的类型,传送信号的电压等。在这一层,数据还没有被组织,仅作为原始的位流或电气电压处理,单位是比特。

(2) 数据链路层(data link layer): 数据链路层负责在两个相邻结点间的线路上,无差错地传送以帧为单位的数据,并进行流量控制。每一帧包括一定数量的数据和一些必要的控制信息。和物理层相似,数据链路层要负责建立、维持和释放数据链路的连接。在传送数据时,如果接收点检测到某帧中所传数据中有差错,就要通知发方重发这一帧。

(3) 网络层(network layer): 为传输层实体提供端到端的交换网络数据传送功能。使得传输层摆脱路由选择、交换方式和拥塞控制等网络传输细节;可以为传输层实体建立、维持和拆除一条或多条通信路径;对网络传输中发生的不可恢复的差错予以报告。

在计算机网络中进行通信的两个计算机之间可能会经过多个数据链路,也可能还要经过很多通信子网。网络层的任务就是选择合适的网间路由和交换节点,确保数据及时传送。网络层将数据链路层提供的帧组成数据包,包中封装有网络层包头,其中含有逻辑地址信息——源站点和目的站点的网络地址。

(4) 传输层(transport layer): 为会话层实体提供透明、可靠的数据传输服务,保证端到端的数据完整性;选择网络层能提供的最适宜的服务;提供建立、维护和拆除传输连接的功能。传输层根据通信子网的特性,最佳地利用网络资源,为两个端系统(也就是源站和目的站)的会话层之间提供建立、维护和断开传输连接的功能,并以可靠和经济的方式传输数据。在这一层,信息的传送单位是报文。

(5) 会话层(session layer): 为彼此合作的表示层实体提供建立、维护和结束会话连接的功能;完成通信进程的逻辑名字与物理名字间的对应;提供会话管理服务。

这一层也可以称为会晤层或对话层。在会话层及以上的高层次中,数据传送的单位不再另外命名,统称为报文。会话层不参与具体的传输,它提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。如服务器验证用户登录便是由会话层完成的。

(6) 表示层(presentation layer): 为应用层进程提供能解释所交换信息含义的一组服务,即将欲交换的数据从适合于某一用户的抽象语法,转换为适合于 OSI 系统内部使用的传送语法。提供格式化的数据表示和转换服务。数据的压缩、解压缩、加密和解密等工作都由表示层负责。

(7) 应用层(application layer): 提供 OSI 用户服务,即确定进程之间通信的性质,以

满足用户需要以及提供网络与用户应用软件之间的接口服务。例如事务处理程序、电子邮件和网络管理程序等。

2. 参考模型的信息流向

如图 5-4 所示, 设 A 系统的用户要向 B 系统的用户传送数据。

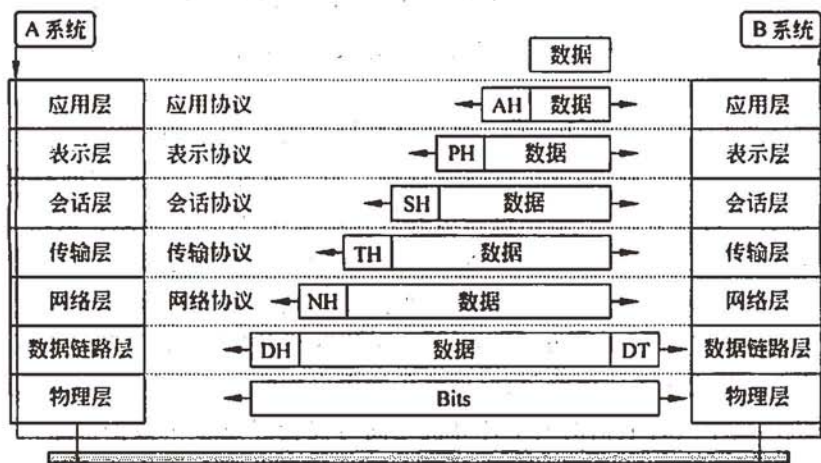


图 5-4 ISO/OSI RM 内的信息流动

A 系统用户的数据先送入应用层。该层给它附加控制信息 AH(头标)后, 送入表示层。表示层对数据进行必要的变换并加头标 PH 后送入会话层。会话层亦加头标 SH 送入传输层。传输层将长报文分段后并加头标 TH 送至网络层。网络层将信息变成报文分组, 并加组号及头标 NH 送数据链路层。数据链路层将信息加上头标和尾标(DH 及 DT)变成帧, 经物理层按位发送到对方(B 系统)。B 系统接收到信息后, 按照与 A 系统相反的动作, 层层剥去控制信息, 最后把原数据传送给 B 系统的用户。可见, 两系统中只有物理层是实通信, 而其余各层均为虚通信。因此, 图 5-4 中只有两物理层间有物理连接, 其余各层间均无连线。

5.3 网络的协议与标准

计算机网络的硬件设备是承载计算机通信的实体, 但它们是怎样有序地完成计算机之间的通信任务的呢? 也就是说, 要共享计算机网络的资源, 以及进行网络中的信息交换, 就需要实现不同系统中的实体的通信。两个实体要想成功地通信, 它们必须具有相同的通信语言, 在计算机网络中称为协议(或规程)。所谓的协议, 指的是网络中的计算机与

计算机进行通信时,为了能够实现数据的正常发送与接收,必须要遵循的一些事先约定好的规程(标准或约定),这些规程中明确规定了通信时的数据格式、数据传送时序以及相应的控制信息和应答信号等内容。下面主要介绍网络的标准、局域网协议与广域网协议。

5.3.1 网络的标准

在网络的标准化方面,有许多标准化机构在工作,如上节介绍的国际标准化组织 ISO、国际电信联盟 ITU、电子工业协会 EIA、电气和电子工程师协会以及因特网活动委员会等。

1. 电信标准

1865 年成立国际电信联盟 ITU(International Telecommunication Union),1947 年 ITU 成为联合国的一个组织,它由 3 部分组成。

- ITU-R: 无线通信部门。ITU-R 的主要工作是确保无线电频率和卫星轨道为所有国家平等、有效和经济地利用,召开世界性和地区性大会来制定无线电法规和地区性协议,起草并通过有关技术、业务和系统的建议。
- ITU-T: 电信标准部门。下设许多研究组,研究组下设专题,从事网络管理、网络维护、业务运营、网络和终端的端对端传输特性、网络总体方面、多媒体业务和系统等等方面的研究。如 Q42/SG VII 专门研究 OSI 参考模型。
- ITU-D: 开发部门。主要宗旨是促进第三世界国家的电信发展。

1953 年至 1993 年,ITU-T 原称为 CCITT(国际电报电话咨询委员会)。CCITT 建议自 1993 年起都打上了 ITU-T 标记。已经公布并使用的最重要的标准如下所述。

1) V 系列

ITU-T 提出的 V 系列标准主要是针对调制解调器的标准。如 V.90 是 56Kbit/s 调制解调器的标准。

2) X 系列

ITU-T 提出的 X 系列标准是应用于广域网的,该系列标准分为两组。

X.1~X.39: 是应用于终端形式、接口、服务设施和设备的标准。最著名的标准是 X.25,它规定了数据包装和传送的协议。

X.40~X.199: 是管理网络结构、传输、发信号等的标准。

2. 国际标准

1946 年成立的国际标准化组织 ISO 负责制定各种国际标准,ISO 有 89 个成员国家,85 个其他成员。ISO 的任务是促进全球范围内的标准化及其有关活动,以利于国际间产品与服务的交流,以及在知识、科学、技术和经济活动中发展国际间的相互合作。例如 ISO 开发了开放式系统互连(Open System Interconnection, OSI)网络结构模型,模型定

义了用于网络结构的 7 个数据处理层。

其他标准化组织如下所示。

(1) ANSI: 美国国家标准研究所, ISO 的美国代表。ANSI 设计了 ASCII 代码集, 它是一种广泛使用的数据通信标准代码。

(2) NIST: 美国国家标准和技术研究所, 美国商业部的标准化机构。

(3) IEEE: 电气和电子工程师协会 (Institute of Electrical and Electronics Engineers)。IEEE 制定了电子工业标准。IEEE 分成一些标准委员会(或工作组), 每个工作组负责标准的一个领域, 工作组 802 制定了网络上的设备如何彼此通信的标准。IEEE 802 标准委员会划分的工作组有 802.1 工作组(协调低档与高档 OSI 模型)、802.2 工作组(涉及逻辑数据链路标准)、802.3 工作组(有关 CSMA/CD 标准在以太网的应用)、802.4 工作组(令牌总线标准在 LAN 中的应用)和 802.5 工作组(设置有关令牌环网络的标准)。

(4) EIA: 电子工业协会(Electronic Industries Association)。最为人熟悉的 EIA 标准之一是 RS-232C 接口, 这一通信接口允许数据在设备之间交换。

值得注意的是, ITU-T 和 ISO 之间有很好的合作和协调。

3. Internet 标准

Internet 的标准特点是自发而非政府干预, 管理松散, 每个分网络均由各自分别管理, 目前已组成了一个民间性质的协会 ISOC(Internet Society), 进行必要的协调与管理, 有一个网络信息中心(NIC)来管理 IP 地址, 保证注册地址的惟一性, 并为用户提供一些文件, 介绍可用的服务。ISOC 设有 Internet 总体管理机构 IAB。

1969 年在 ARPANET 时代就开始发布请求评议 RFC(Request for Comments), 至今已超过 3000 个。

5.3.2 局域网协议

IEEE 局域网标准委员会对局域网的定义为: “局域网络中的通信被限制在中等规模的地理范围内, 如一所学校; 能够使用具有中等或较高数据速率的物理信道, 且具有较低的误码率; 局域网络是专用的, 由单一组织机构所使用。”局域网技术由于具有规模小、组网灵活和结构规整的特点, 所以极易形成标准。事实上, 局域网技术在所有计算机网络技术中是标准化程度最高的。国际电子电气工程师协会 IEEE 早在 20 世纪 70 年代就制定了 3 个局域网标准: IEEE 802.3(CSMA/CD, 以太网)、IEEE 802.4(Token Bus, 令牌总线)和 IEEE 802.5(Token Ring, 令牌环)。由于它已被市场广泛接受, 所以 IEEE 802 系列标准已被 ISO 采纳为国际标准。而且, 随着网络技术的发展, 又出现了 IEEE 802.7(FDDI)、IEEE 802.3u(快速以太网)、IEEE 802.12(100VG-AnyLAN)和 IEEE 802.3z

(千兆以太网)等新一代网络标准。

一个局域网的基本组成主要有网络服务器、网络工作站、网络适配器和传输介质。这些设备在特定网络软件支持下完成特定的网络功能。决定局域网特性的主要技术有 3 个方面：用以传输数据的传输介质；用以连接各种设备的拓扑结构；用以共享资源的介质访问控制方法。它们在很大程度上决定了传输数据的类型、网络的响应时间、吞吐量和利用率，以及网络应用等各种网络特性。不同的局域网协议最重要的区别是介质访问控制方法，它对网络特性具有十分重要的影响。

1. LAN 模型

ISO/OSI 的 7 层参考模型本身并不是一个标准，在制定具体网络协议和标准时，要依 OSI 参考模型作为“参照基准”，并说明与该“参照基准”的对应关系。在 IEEE802 局域网(LAN)标准中，只定义了物理层和数据链路层两层，并根据 LAN 的特点，把数据链路层分成逻辑链路控制 LLC(logical link control)子层和介质访问控制 MAC(media access control)子层，还加强了数据链路层的功能，把网络层中的寻址、排序、流控和差错控制等功能放在 LLC 子层来实现。图 5-5 为 LAN 协议的层次以及与 OSI 参考模型的对应关系。

1) 物理层

和 OSI 物理层的功能一样，主要处理在物理链路上发送、传递和接收非结构化的比特流，包括对带宽的频道分配和对基带的信号调制，建立、维持和撤销物理链路，处理机械的、电气的和过程的特性。其特点是可以采用一些特殊的通信媒体，在信息组成的格式上可以有多种。

2) 介质访问控制层 MAC

主要功能是控制对传输介质的访问，MAC 与网络的具体拓扑方式以及传输介质的类型有关，主要是介质的访问控制和对信道资源的分配。MAC 层还实现对帧的寻址和识别，完成帧检测序列的产生和检验等功能。

3) 逻辑链路控制层 LLC

可提供两种控制类型，即面向连接服务和非连接服务。其中，面向连接服务能够提供可靠的信道。逻辑链路控制层提供的主要功能是数据帧的封装和拆除，为高层提供网络服务的逻辑接口，能够实现差错控制和流量控制。

在计算机网络体系结构中，最具代表性和权威的是 ISO 的 OSI/RM 和 IEEE 的 802 协议。OSI 是设计和实现网络协议标准的最重要的参考模型和依据，而 IEEE 802 则制定

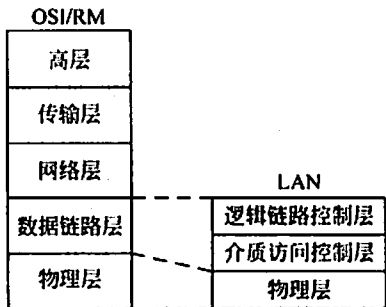


图 5-5 LAN 层次与 OSI/RM 的对应关系

了一系列具体的局域网标准,并不断地增加新的标准。它们之间的关系如图 5-6 所示。

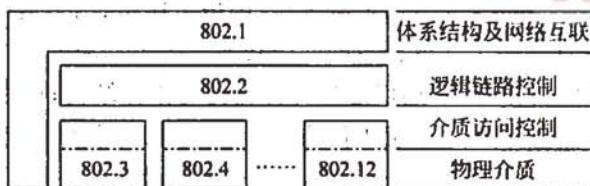


图 5-6 IEEE 802 标准系列间的关系

2. 以太网(IEEE 802.3 标准)

以太网技术可以说是局域网技术中历史最悠久和最常用的一种。它采用的“存取方法”是带冲突检测的载波监听多路访问协议(CSMA/CD, Carrier-Sense Multiple Access with Collision Detection)技术。

目前以太网主要包括 3 种类型: IEEE 802.3 中定义的标准局域网,速度为 10Mb/s,传输介质为细同轴电缆;IEEE 802.3u 中定义的快速以太网,速度为 100Mb/s,传输介质为双绞线;IEEE 802.3z 中定义的千兆以太网,速度为 1000Mb/s,传输介质为光纤或双绞线。

1) 介质访问技术

IEEE 802.3 所使用的介质访问协议 CSMA/CD 是让整个网络上的主机都以竞争的方式来抢夺传送数据的权力。工作过程为:首先侦听信道,如果信道空闲则发送。如果信道忙,则继续侦听,直到信道空闲时立即发送。开始发送后再进行一段时间的检测,方法是边发送边接收,并将收、发信息相比较,若结果不同,表明发送的信息遇到碰撞,于是立即停止发送,并向总线上发出一串阻塞信号,通知各站信道上冲突已发生。已发出信息的各站收到阻塞信号后,等待一段随机时间。等待时间最短的站将重新获得信道,可重新发送。

在 CSMA/CD 中,当检测到冲突并发出阻塞信号后,为了降低再次冲突的概率,需要等待一个退避时间。退避算法有许多种,常用的一种通用退避算法称为二进制指数退避算法。

2) IEEE 802.3——10Mb/s 以太网

IEEE 802.3——10Mb/s 以太网定义过 10BASE5、10BASE2、10BASE-T 和 10BASE-F 等几种(需要注明的是,其中 10BASE-T 与 10BASE-F 的最后项就是以线缆类型进行命名的,其中 T 代表双绞线, F 代表光纤)。10BASE5 标准是最早的媒体规范,它使用阻抗为 50Ω 的同轴粗缆。但由于同轴粗缆的缆线直径大,所以比较笨重,不易铺设。10BASE2 标准是为建立一个比 10BASE5 更廉价的局域网制定的,它使用阻抗为 50Ω 的

同轴细缆,惟一的差别就是它使得每两个节点间的距离限制从 500m 降为 185m。10BASE-T 标准是一个使用非屏蔽双绞线为传输介质的标准,所要用到的非屏蔽双绞线只需 3 类线标准即可满足要求,是一个成功的标准。10BASE-F 标准充分利用了新兴媒体光纤的距离长、传输性能好的优点,大大改进了以太网技术。

3) IEEE 802.3u——100Mb/s 快速以太网

随着计算机技术的不断发展,10Mb/s 的网络传输速度实在无法满足日益增大的需求。IEEE 802.3u 充分考虑到了向下兼容性:采用非屏蔽双绞线(或屏蔽双绞线、光纤)作为传输媒介,采用与 IEEE 802.3 一样的介质访问控制层——CSMA/CD。IEEE 802.3u 常称为快速以太网。根据实现的介质不同,快速以太网可以分为 100BaseTX、100BaseFX 和 100BaseT4 3 种。

100BaseTX 用两对 5 类非屏蔽双绞线(UTP),或者 1 类、2 类屏蔽双绞线(STP)作为传输媒介,来实现传输速度为 100Mb/s 的网络,最多支持两个中继器。100BaseFX 是两束多模光纤上的标准,在没有中继设备的网络中最大距离为 400m。100BaseT4 利用 10Mb/s 网络中使用的 3 类线有两对空着未用的特点,使用 4 对 3 类非屏蔽双绞线提供传输速度为 100Mb/s 网络。

4) IEEE 802.3z——1000Mb/s 千兆以太网

IEEE 802.3z 对介质访问控制(MAC)层规范进行了重新定义,以维持适当的网络传输距离,介质访问控制方法仍采用 CSMA/CD 协议,并且重新制定了物理层标准,使之能提供 1000Mb/s 的原始带宽。因此它仍是一种共享介质的局域网,发送到网上的信号是广播式的,接收站根据地址接收信号。网络接口硬件能监听线路上是否已存在信号,以避免冲突,或在没有冲突时重发数据。

在物理层,千兆以太网支持 3 种传输介质。光纤系统:支持多模光纤和单模光纤系统,多模光纤的工作距离为 500m,单模光纤的工作距离为 2000m;宽带同轴电缆系统:其传输距离为 25m;5 类 UTP 电缆:其传输距离为 100m,链路工作模式为半双工。

千兆以太网采用以交换机为中心的星形拓扑结构,将网络核心部件挂接到千兆以太网交换机上。主要用于交换机与交换机之间或者交换机与企业超级服务器之间的高速网络连接。

3. 令牌环网(IEEE 802.5)

令牌环是环型网中最普遍采用的介质访问控制,它适用于环形网络结构的分布式介质访问控制,其流行性仅次于以太网。令牌环网的传输介质虽然没有明确定义,但主要基于屏蔽双绞线、非屏蔽双绞线两种,拓扑结构可以有多种:环型(最典型)、星型(采用得最多)以及总线型(一种变形),编码方法为差分曼彻斯特编码。

IEEE 802.5 的介质访问使用的是令牌环控制技术,工作过程为:首先,令牌环网在

网络中传递一个很小的帧,称为“令牌”,只有拥有令牌环的工作站才有权力发送信息;令牌在网络上依此顺序传递;当工作站要发送数据时,它应等待捕获一个空令牌,然后将要发送的信息附加到后边,发往下一站,如此直到目标站,然后将令牌释放;如果工作站要发送数据,经过的令牌不为空时,则需等待令牌释放。

当信息帧绕环网通过各站时,各站都要将帧的目的地址与本站地址进行比较。如果地址一致,说明是发送给本站的,此时可将帧拷贝到本站的接收缓冲器中,同时将帧送回环上,使帧继续沿环传送;如果地址不符合,只需简单地将信息帧重新送到环上即可。

4. FDDI(光纤分布式数据接口)

FDDI(Fiber Distributed Data Interface)类似于令牌环网协议,它用光纤作为传输介质,数据传输速率可达到 100Mb/s,环路长度可扩展到 200km,连接的站点数可以达到 1000 个。FDDI 采用一种新的编码技术,称为 4B/5B 编码。它每次对 4 位数据进行编码,每 4 位数据编码成 5 位符号,用光信号的存在与否来代表 5 位符号中的每一位是 1 还是 0。

光纤中传送的是光信号,有光脉冲表示 1,无光脉冲表示 0。这种简单编码的缺点是没有同步功能。在使用同轴电缆或双绞线作为传输介质的局域网中,通常采用曼彻斯特编码方式。它利用中间的跳变作为同步信号,对每一位数据单元产生两次瞬变,因而使带宽的利用率降低。在 5 比特编码的 32 种组合中,实际只使用了 24 种,其中的 16 种用做数据,其余 8 种用做控制符号(如帧的起始和结束符号等)。在 4B/5B 编码中,其中 5 位码中的“1”码至少为两位,按 NRZI 编码原理,信号中就至少有两次跳变,因此接收端可得到足够的同步信息。

FDDI 采用双环体系结构,两环上的信息反方向流动。双环中的一环称为主环,另一环称为次环。在正常情况下,主环传输数据,次环处于空闲状态。双环设计的目的是提供高可靠性和稳定性。FDDI 定义的传输介质有单模光纤和多模光纤两种。FDDI 与网络设备有多种连接方法:单连接站点(SAS)、双连接站点(DAS)和集线器三种类型的设备。

5.3.3 广域网协议

广域网通常是指覆盖范围大,传输速率低,以数据通信为主要目的的数据通信网。随着信息技术的迅速发展,很多国家的数据通信业务的增长率已大大提高,特别是国际互联网的普及促进了数据通信网技术的发展。

在地域分布很远、很分散,以致于无法用直接连接来接入局域网的场合,广域网(WAN)通过专用的或交换式的连接把计算机连接起来。这种广域连接可以通过公众网建立的,也可以是通过服务于某个专门部门的专用网建立起来的。相对来说,广域网显

得比较错综复杂。目前,主要用于广域传输的协议比较多,如 PPP(点对点协议)、DDN、ISDN(综合业务数字网)、FR(帧中继)和 ATM(异步传输模式)等。

1. 点对点协议 PPP

PPP 点对点协议主要用于“拨号上网”这种广域连接模式。它的优点在于简单,具备用户验证能力,可以解决 IP 分配等。它主要通过拨号或专线方式建立点对点连接,发送数据,使其成为各种主机、网桥和路由器之间的一种通用的简单连接解决方案。

家庭拨号上网就是通过 PPP 在用户端和运营商的接入服务器之间建立通信链路。目前,宽带接入正呈取代拨号上网的趋势。在宽带接入技术日新月异的今天,PPP 也衍生出新的应用。典型的应用是在 ADSL(非对称数据用户线,Asymmetrical Digital Subscriber Line)接入方式当中,PPP 与其他的协议共同派生出了符合宽带接入要求的新的协议,如 PPPoE(PPP over Ethernet)和 PPPoA(PPP over ATM)。

利用以太网(Ethernet)资源,在以太网上运行 PPP 并进行用户认证的接入方式称为 PPPoE。PPPoE 既保护了用户方的以太网资源,又完成了 ADSL 的接入要求,是目前 ADSL 接入方式中应用最广泛的技术标准。同样,在 ATM 网络上运行 PPP 协议来管理用户认证的方式称为 PPPoA。它与 PPPoE 的原理相同,作用也相同;不同的是它在 ATM 网络上,而 PPPoE 在以太网网络上运行,所以要分别适应 ATM 标准和以太网标准。

2. 数字用户线 xDSL

xDSL 是各种数字用户线的统称。为满足各种宽带通信业务的需要,目前还有 DSL 技术和产品,其中包括 ADSL(Asymmetric DSL)即不对称数字用户线、SDSL(single pair DSL)单对线数字用户环路、IDSL(ISDN DSL)即 ISDN 用的数字用户线、RADSL(rate adaptive DSL)速率自适应非对称型数字用户线以及 VDSL(very high Bit rate DSL)甚高速数字用户线等。

ADSL(Asymmetrical Digital Subscriber Line)是研制最早,发展较快的一种。它是在一对铜双绞线上,为用户提供上、下行非对称的传输速率(即带宽)。ADSL 接入服务能做到较高的性能价格比。ADSL 接入技术较其他接入技术具有独特的技术优势:它的速率可达到上行 1M/下行 8M,速度非常快;另外,使用 ADSL 上网不需要占用电话线路,在电话和上网互不干扰的同时,大大节省了普通上网方式的话费支出;独享带宽,安全可靠;安装快捷方便;价格实惠。它把线路按频段分成语音、上行和下行 3 个信道,语音和数据可共用 1 对线。ADSL 特别适合于像 VOD 业务及 Internet 和多媒体业务的应用。ADSL 一般采用 CAP 和 DMT 两种线路编码调制技术。传输距离与线径、速率有关,一般在 3km 以上。因此,ADSL 是一种很有发展前途的数字接入技术。ADSL 技术作为一种宽带接入方式,可以为用户提供宽带网的所有应用业务。采用各种拨号方式上网的用

户将逐步过渡到 ADSL 宽带接入方式。ADSL 在宽带接入中已经扮演着越来越重要的角色。

对于个人用户,在现有电话线上安装 ADSL,只须在用户端安装一台 ADSL MODEM 和一只分离器,用户线路不用做任何改动,极其方便。数据线路为:PC—ADSL Modem—分离器—入户接线盒—电话线—DSL 接入复用器—ATM/IP 网络;语音线路为:话机—分离器—入户接线盒—电话线—DSL 接入复用器—交换机。

对于企业用户,可在现有电话线上安装 ADSL 和分离器,连接 hub 或 switch。数据线路为:PC—以太网(hub 或 switch)—ADSL 路由器—分离器—入户接线盒—电话线—DSL 接入复用器—ATM/IP 网络;语音线路为:话机—分离器—入户接线盒—电话线—DSL 接入复用器—交换机。

3. 数字专线 DDN

数字数据网(Digital Data Network)是采用数字传输信道传输数据信号的通信网,可提供点对点、点对多点的透明传输的数据专线出租电路,为用户传输数据、图像和声音等信息。数字数据网以光纤为中继干线网络,组成 DDN 的基本单位是节点,节点间通过光纤连接,构成网状的拓扑结构。

DDN 专线就是市内或长途的数据电路,电信部门将它们出租给用户做资料传输使用后,它们就变成了用户专线,直接进入电信的 DDN 网络,因为这种电路采用固定连接的方式,不需经过交换机房,所以称之为固定 DDN 专线。DDN 专线不仅需要铺设专用线路(DDN 的客户端需要一个称为 DDN MODEM 的 CSU/DSU 设备,以及一个路由器),从用户端进入主干网络,而且需要付电信月租费、网络使用费和电路租用费等。其优势是网络传输速率高、时延小、质量好、网络透明度高、可支持任何规程以及安全可靠。

4. 综合业务数字网 ISDN

综合业务数字网是建立在数字电话网基础上的网络。它能提供端到端的数字连接,将声音、数据、图像和传真等不同业务综合在一个网络内进行传送和处理。客户只需一条普通的电话线,通过网络接口(NT)及相应的终端设备,拨通号码后就可以达到既能上网又能打电话的目的。中国电信称其为“一线通”。

窄带综合业务数字网向用户提供基本速率(2B+D,144Kb/s)和一次群速率(30B+D,2Mb/s)两种接口。基本速率接口(BRI)是一条标准的 ISDN 用户电路,它包含两个 B 信道和一个 D 信道,B 信道一般用来传输话音、数据和图像,D 信道用来传输信令或分组信息,各个 B 信道均能够以 64Kb/s 的速率传输数据,D 信道能够以 16Kb/s 的速率传输,支持的吞吐量达 144Kb/s 分组交换数据。一次群速率接口(PRI)具有 30 个独立的或组合的 64Kb/s 的 B 信道和一个 64Kb/s 的 D 信道,提供高达 2.048Mb/s 的传输速率。

ISDN 可以提供电话、传真、可视图文及数据通信等多种业务。其优点是它能够实现高可靠性及高质量的通信,使用方便,费用低廉。

5. 帧中继 FR

帧中继(Frame Relay)是在用户网络接口之间提供用户信息流的双向传送,并保持顺序不变的一种承载业务。用户信息以帧为单位进行传输,并对用户信息流进行统计复用。帧中继是综合业务数字网标准化过程中产生的一种重要技术,它是在数字光纤传输线路逐渐代替原有的模拟线路,用户终端智能化的情况下,由 X.25 分组交换技术发展起来的一种传输技术。

帧中继是一种基于可变帧长的数据传输网络。在传输过程中,网络内部可以采用“帧交换”,即以帧为单位进行传送,也可采用“信元交换”(每个信元 53B)为单位进行传送。

帧中继提供一种简单的面向连接的虚电路分组服务,包括交换虚电路连接和永久虚电路连接。帧中继的优点有:降低网络互联费用,简化网络功能,提高网络性能,采用国际标准,各厂商产品相互兼容。

在实际应用中,帧中继主要适用于以下几种情况:

- 当客户的带宽需求为 64Kb/s~2 Mb/s,而参与通信的节点多于两个的时候,使用帧中继是一种较好的解决方案。
- 当通信距离较长时,帧中继的高效性使用户可以享有较好的经济性。
- 当客户传送的数据突发性较强时,由于帧中继具有动态带宽分配的功能,选用帧中继可以有效地处理突发性数据。

6. 异步传输模式 ATM

异步传输模式 ATM(asynchronous transfer mode)是 B-ISDN 的关键核心技术,它是一种面向分组的快速分组交换模式,使用异步时分复用技术,将信息流分割成固定长度的信元。使用统一的信息单位能比较容易地实现各种信息流混合在一起的多媒体通信,并能根据业务类型、速率的需求动态地分配有效容量。ATM 能够根据需要改变传送速率,对高速信息传递频次高,而对低速信息传递频次低,按照统计复用的原理进行传输和交换。故 ATM 完全可以用单一的交换方式,灵活有效地支持频带分布范围极广的各种业务。

在 ATM 网络中,数据以定长的信元为单位进行传输,信元由信元头和信元体构成,每个信元 53B,其中信元头 5B,信元体 48B。

ATM 的参考模型由 4 层构成,它们分别是:用户层、ATM 适配层、ATM 层和物理层。图 5-7 给出了简化后的 ATM 的参考模型。

1) 用户层

由用户平面、控制平面和管理平面组成。其各自的功能如下:



图 5-7 ATM 的参考模型

(1) 用户平面：指用户要求的应用、协议及服务。如通常的数据协议、语音和视频应用等。

(2) 控制平面：包含连接建立、维护和拆除等有关功能。

(3) 管理平面：负责层次和平面的协调，通过层次管理支持用户平面和管理平面。

2) ATM 适配层

负责将用户层的信息转换成 ATM 网络可用的格式。等用户层把较长的数据分组交给 ATM 适配层后，ATM 适配层按规定的长度将数据分成若干信元体，再传给 ATM 层。

3) ATM 层

基本功能是负责信元生成。它不管信元体的内容，只为信元体生成信元头并附加到信元体，以形成标准的信元格式。跨越 ATM 层到物理层的信息单元只能是 53B 的信元。

4) 物理层

负责对信元进行编码，并将其交给物理介质。它汇集了各种被认可的物理线路协议，如 SDH(STM-1、STM-4 和 STM-16)和同步光纤网等。

ATM 连接有两种类型，即永久虚电路(PVC)和交换虚电路(SVC)。

- 永久虚电路 PVC：其连接由管理员建立和拆除。
- 交换虚电路 SVC：通过终端系统和 ATM 网络以及 ATM 网络内部的信令协议的操作来建立和拆除。

虽然这里将 ATM 技术划在广域网部分来介绍，但 ATM 却是一种可以将局域网功能、广域网功能、语音、视频和数据集成为一个统一的协议设计。正是它的高度统一性和良好的可扩展性，给计算机网络技术掀开了新的一页。它具有的特点：速度快，支持 622Mb/s 的传输速率；可扩展性好；较高的传输质量 QoS；提供端到端解决方案的潜力，这意味着它的应用可以从桌面到局域网，一直延伸到广域网。

ATM 技术的适用范围为：由于 ATM 技术提供了基于专用带宽的设计和数据优先级设计，使得它特别适合多媒体和视频应用；ATM 技术具有良好的扩展能力以及高性能

的网络传输能力,适合构架骨干网;ATM 具有高性能的无缝地集成广域网和局域网的能力,所以被广泛地应用于广域网建设中。ATM 主要缺点是成本较高,不适合于小网络。

7. X.25 协议

X.25 是在公用数据网上以分组方式进行操作的 DTE(数据终端设备)和 DCE(数据通信设备)之间的接口。X.25 只是对公用分组交换网络的接口规范说明,并不涉及网络内部实现。它是面向连接的,支持交换式虚电路和永久虚电路。

X.25 在本地 DTE 和远程 DTE 之间提供一个全双工、同步的透明信道,并定义了 3 个相互独立的控制层:物理层、链路层和分组层,它们分别对应于 ISO/OSI 的物理层、链路层和网络层,如图 5-8 所示。

物理层接口指 DTE 和网络之间的线路连接,X.25 指出可采用 V.24、V.35、G.703 和 X.21 等接口;链路层逻辑接口采用链路层协议,负责 DTE 和 DCE 之间的初始化和校验,并控制物理链路上的数据传输,用户数据以信息帧格式在 DTE 和 DCE 之间传送;分组层逻辑接口描述了呼叫的建立、保持和拆除的过程,以及数据和控制信息在分组中的格式。默认的数据分组长度是 128 个字节。



图 5-8 X.25 层次模型

5.3.4 Internet 协议

TCP/IP 作为 Internet 的核心协议,通过近二十多年的发展已日渐成熟,并被广泛应用于局域网和广域网中,目前已成为事实上的国际标准。作为一个最早的、也是迄今为止发展最为成熟的互联网络协议系统,TCP/IP 包含许多重要的基本特性,这些特性主要表现在 5 个方面:逻辑编址、路由选择、域名解析、错误检测和流量控制以及对应用程序的支持等。

- 逻辑编址:每一块网卡在出厂时就由厂家分配了一个独一无二的永久性的物理地址。在 Internet 中,需为每台连入因特网的计算机分配一个逻辑地址,这个逻辑地址称为 IP 地址。一个 IP 地址可以包括:一个网络 ID 号,用来标识网络;一个子网络 ID 号,用来标识网络上的一个子网;另外,还有一个主机 ID 号,用来标识子网络上的一台计算机。这样,通过这个分配给某台计算机的 IP 地址,就可以很快地找到相应的计算机。
- 路由选择:TCP/IP 中包含专门用于定义路由器如何选择网络路径的协议,即 IP 数据包的路由选择协议。

- 域名解析：虽然 TCP/IP 采用 32 位的 IP 地址，但考虑到用户记忆的方便，专门设计了一种方便的字母式地址结构，称为域名或 DNS(域名服务)名字。将域名映射为 IP 地址的操作，称为域名解析。域名具有较稳定的特点，而 IP 地址则较易发生变化。
- 错误检测与流量控制：TCP/IP 具有确保数据信息在网络上可靠传递的特性，这些特性包括数据信息传输的错误检测(保证到达目的地的数据信息没有发生变化)，确认已传递的数据信息已被成功地接收，监测网络系统中的信息流量，防止出现网络拥塞。

1. TCP/IP 分层模型

协议是对数据在计算机或设备之间传输时的表示方法进行定义和描述的标准。协议规定了传输数据、检测错误以及传送确认信息等内容。TCP/IP 是个协议族，它包含多种协议。ISO/OSI 模型、TCP/IP 的分层模型及协议的对比如图 5-9 所示。

ISO/OSI 模型			TCP/IP 协议			TCP/IP 模型	
应用层	文件传输 协议 FTP	远程登录 协议 Telnet	电子邮件 协议 SMTP	网络文件 服务协议 NFS	网络管理 协议 SNMP	应用层	
表示层							
会话层							
传输层	TCP			UDP		传输层	
网络层	IP		ICMP		ARP RARP	网际层	
数据链路层	Internet	FDDI	Token-Ring/ IEEE 802.5	ARCnet	PPP/SLIP	网络接口层	
物理层	IEEE 802.3					硬件层	

图 5-9 TCP/IP 模型与 OSI 模型的对比

从图 5-9 可知，TCP/IP 分层模型由 4 个层次构成，即应用层、传输层、网际层和网络接口层。其各层的功能简述如下所示。

(1) 应用层，应用层处在分层模型的最高层，用户调用应用程序来访问 TCP/IP 互联网络，以享受网络上提供的各种服务。应用程序负责发送和接收数据。每个应用程序可以选择所需要的传输服务类型，并把数据按照传输层的要求组织好，再向下层传送，包括独立的报文序列和连续字节流两种类型。

(2) 传输层，传输层的基本任务是提供应用程序之间的通信服务。这种通信又叫端到端的通信。传输层既要系统地管理数据信息的流动，还要提供可靠的传输服务，以确保数据准确而有序地到达目的地。为了达到这个目的，传输层协议软件需要进行协商，让接收方回送确认信息及让发送方重发丢失的分组。在传输层与网际层之间传递的对象是传

输层分组。

(3) 网际层,网际层又称 IP 层,主要处理机器之间的通信问题。它接收传输层的请求,传送某个具有目的地址信息的分组。该层主要完成:

- 把分组封装到 IP 数据报(IP Datagram)中,填入数据报的首部(也称为报头),使用路由算法选择,把数据报直接送到目标机或把数据报发送给路由器,然后,再把数据报交给下面的网络接口层中对应的网络接口模块。
- 处理接收到的数据报,检验其正确性。使用路由算法来决定是在本地进行处理,还是继续向前发送。如果数据报的目标机处于本机所在的网络,该层软件就把数据报的报头剥去,再选择适当的传输层协议软件来处理这个分组。
- 适时发出 ICMP 的差错和控制报文,并处理收到的 ICMP 报文。

(4) 网络接口层 网络接口层又称数据链路层,处于 TCP/IP 协议层之下,负责接收 IP 数据报,并把数据报通过选定的网络发送出去。该层包含设备驱动程序,也可能是一个复杂的使用自己的数据链路协议的子系统。

2. 网络接口层协议

TCP/IP 协议不包含具体的物理层和数据链路层,只定义了网络接口层与物理层的接口规范。这个物理层可以是广域网,如 X.25 公用数据网,可以是局域网,如 Ethernet、Token-Ring 和 FDDI 等。任何物理网络,只要按照这个接口规范开发网络接口驱动程序,都能够与 TCP/IP 协议集成起来。网络接口层处在 TCP/IP 协议的最底层,主要负责管理为物理网络准备数据所需的全部服务程序和功能。

3. 网际层协议——IP 协议

网际层是整个 TCP/IP 协议族的重点之一。网际层定义的协议除了 IP 外,还有 ICMP、ARP 和 RARP 等几个重要的协议。

IP 所提供的服务通常被认为是无连接的(connectionless)和不可靠的(unreliable)。事实上,在网络性能良好的情况下,IP 传送的数据能够完好无损地到达目的地。所谓无连接的传输,是指没有确定目标系统是否已做好接收数据准备之前就发送数据。与此相对应的就是面向连接的(connection oriented)传输(如 TCP),在该类传输中,源系统与目的系统的应用层数据开始传送之前需要进行 3 次握手。不可靠的服务是指目的系统不对成功接收的分组进行确认,IP 只是尽可能地使数据传输成功。但只要需要,上层协议必须实现用于保证分组成功传输的附加服务。

由于 IP 只提供无连接、不可靠的服务,所以把差错检测和流量控制之类的服务授权给其他各层协议,这正是 TCP/IP 能够高效率工作的一个重要保证。这样,可以根据传送数据的属性来确定所需的传送服务以及客户应该使用的协议。例如,传送大型文件的 FTP 会话就需要面向连接的、可靠的服务(因为如果稍有损坏,就可能导致整个文件无法

使用)。

IP 协议的主要功能包括:将上层数据(如 TCP 和 UDP 数据)或同层的其他数据(如 ICMP 数据)封装到 IP 数据报中;将 IP 数据报传送到最终目的地;为了使数据能够在链路层上进行传输,对数据进行分段;确定数据报到达其他网络中的目的地的路径。

IP 协议软件的工作流程:当发送数据时,源计算机上的 IP 协议软件必须确定目的地是在同一个网络上,还是在另一个网络上。IP 通过执行这两项计算并对结果进行比较,才能确定数据到达的目的地。如果两项计算的结果相同,则数据的目的地可以确定为本地,否则,目的地应为远程的其他网络。如果目的地在本地,那么 IP 协议软件就启动直达通信;如果目的地是远程计算机,那么 IP 必须通过网关(或路由器)进行通信。在大多数情况下,这个网关应当是缺省网关。当源 IP 完成了数据报的准备工作时,它就将数据报传递给网络访问层,网络访问层再将数据报传送给传输介质,最终完成数据帧发往目的计算机的过程。

当数据抵达目的计算机时,网络访问层首先接收该数据。网络访问层要检查数据帧有无错误,并将数据帧送往正确的物理地址。假如数据帧到达目的地时正确无误,网络访问层便从数据帧的其余部分中提取数据有效负载(Payload),然后将它一直传送到帧层次类型域指定的协议。在这种情况下,可以说数据有效负载已经传递给了 IP。

有关 IP 地址的具体情况参见 5.5 节的 Internet 地址。

4. ARP 和 RARP 协议

地址解析协议 ARP(Address Resolution Protocol)及反地址解析协议 RARP 是驻留在网际层中的另一个重要协议。ARP 的作用是将 IP 地址转换为物理地址,RARP 的作用是将物理地址转换为 IP 地址。网络中的任何设备,包括主机、路由器和交换机等均有惟一的物理地址,该地址通过网卡给出,每个网卡出厂后都有不同的编号,这意味着用户所购买的网卡都有惟一的物理地址。另一方面,为了屏蔽底层协议及物理地址上的差异,IP 协议又使用了 IP 地址。因此,在数据传输过程中,必须对 IP 地址与物理地址进行相互转换。

用 ARP 进行 IP 地址到物理地址转换的过程为:当计算机需要与任何其他计算机进行通信时,首先需要查询 ARP 高速缓存。如果 ARP 高速缓存中存在这个 IP 地址,便使用与它对应的物理地址,直接将数据报发送给相应的物理网卡。如果 ARP 高速缓存中没有该 IP 地址,那么 ARP 便在局域网上以广播方式发送一个 ARP 请求包。如果局域网上的 IP 地址与某台计算机中的 IP 地址一致,那么该计算机便生成一个 ARP 应答信息,信息中包含对应的物理地址。ARP 协议软件将 IP 地址与物理地址的组合添加到它的高速缓存中,这时即可开始数据通信。

RARP 负责物理地址到 IP 地址的转换。这主要用于无盘工作站上。网络上无盘工

作站的网卡上有自己的物理地址,但无 IP 地址,因此必须有一个转换过程。为了完成这个转换过程,网络中应有一个 RARP 服务器,网络管理员事先必须把网卡上的 IP 地址和相应的物理地址存储到 IP RARP 服务器的数据库中。

5. 网际层协议——ICMP 协议

Internet 控制信息协议(Internet Control Message Protocol, ICMP)是网际层的另一个比较重要的协议。由于 IP 协议是一种尽力传送的通信协议,即传送的数据报可能会丢失、重复、延迟或乱序,所以 IP 协议需要有一种在发生差错时报告的机制。ICMP 就是一个专门用于发送差错报文的协议。ICMP 定义了 5 种差错报文(源抑制、超时、目的不可达、重定向和要求分段)和 4 种信息报文(回应请求、回应该答、地址屏蔽码请求和地址屏蔽码应答)。IP 在需要发送一个差错报文时要使用 ICMP,而 ICMP 却也是利用 IP 来传送报文的。ICMP 是让 IP 更加稳固、有效的一种协议,它使 IP 传送机制变得更加可靠。而且 ICMP 还可以用于测试互联网,以得到一些有用的网络维护和排错的信息。例如,著名的 ping 工具就是利用 ICMP 报文进行目标可达性测试。

6. 传输层协议——TCP 协议

传输控制协议(Transmission Control Protocol, TCP)是整个 TCP/IP 协议族中最重要的协议之一。它在 IP 协议提供的不可靠数据服务的基础上,为应用程序提供了一个可靠的、面向连接的和全双工的数据传输服务。

TCP 协议是如何实现可靠性的呢?最主要和最重要的是 TCP 采用了一种重发(retransmission)技术。具体来说,在 TCP 传输过程中,发送方启动一个定时器,然后将数据包发出,当接收方收到了这个信息就给发送方一个确认(acknowledgement)信息。如果发送方在定时器到时之前没收到确认信息,就重新发送这个数据包。

利用 TCP 协议在源主机和目的主机之间建立和关闭连接操作时,均需要通过 3 次握手来确认建立和关闭是否成功。3 次握手方式如图 5-10 所示,它通过“序号/确认号”使它们的序号达成同步,从而确保系统正常工作。TCP 建立连接的 3 次握手过程如下所示。

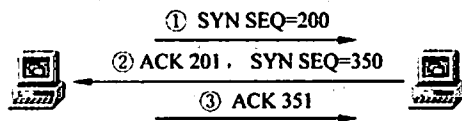


图 5-10 TCP 建立连接的“3 次握手”过程

① 源主机发送一个 SYN(同步)标志位为 1 的 TCP 数据包,表示想与目标主机进行通信,并发送一个同步序列号(如: SEQ=200)进行同步。

② 目标主机愿意进行通信,则响应一个确认(ACK 位设置为 1)。并以下一个序列号

为参考进行确认(如: SEQ=201)。

③ 源主机以确认来响应目标主机的 TCP 包。这个确认中包括它想要接收的下一个序列号(该帧可以含有发送的数据)。至此连接建立完成。

同样,关闭连接也进行三次握手。

7. 传输层协议——UDP 协议

用户数据报协议(User Datagram Protocol, UDP)是一种不可靠的、无连接的协议,不能保证应用程序进程间的通信。与同样处在传输层的面向连接的 TCP 相比较,UDP 是一种无连接的协议,它的错误检测功能要弱得多。可以这样说,TCP 有助于提供可靠性,而 UDP 则有助于提高传输的高速率。例如,必须支持交互式会话的应用程序(如 FTP 等)往往使用 TCP 协议,而自己进行错误检测或不需要错误检测的应用程序(如 DNS 和 SNMP 等)则往往使用 UDP。

UDP 协议软件的主要作用就是将 UDP 消息展示给应用层,它并不负责重新发送丢失的或出错的数据消息,不会对接收到的无序 IP 数据报重新排序,不消除重复的 IP 数据报,不对已收到的数据报进行确认,也不负责建立或终止连接。而这些问题是由使用 UDP 进行通信的应用程序负责处理的。

TCP 协议虽然提供了一种可靠的数据传输服务,但它是牺牲通信量来实现的。也就是说,为完成同样一个任务,TCP 会需要更多的时间和通信量。这在网络不可靠的情况下,通过牺牲一些时间来达到网络的可靠是值得的。但当网络十分可靠的时候,它仍然通过浪费带宽来保证可靠性,这时 UDP 十分小的通信量浪费则占据优势。

在某些应用中,每个数据的传输可靠性并不十分重要,重要的却是整个网络的传输速度。例如语音传输,如果其中的一个包丢失了,重发也没有意义,因为这个语音数据已经失效了,否则会听到前一时间的话音。

8. 应用层协议

在计算机网络的广泛应用中,人们已经有了许多相同的基本应用需求。为了让不同平台的计算机能够通过计算机网络提供一些基本的相同的服务,也就应运而生了一系列应用级的标准,实现这些应用标准的专用协议被称为应用级协议。相对于 OSI 参考模型来说,它们处于较高的层次结构,所以也称为高层协议。应用层的协议有 NFS、Telnet、SMTP、DNS、SNMP 和 FTP 等,其详细情况将在 5.5 节 Internet 服务中介绍。

5.4 构建网络

构建一个实际的网络是一项复杂的系统工程,整个过程一般分为网络规划、网络设计、工程施工和系统运行维护等阶段。本节主要介绍构建网络的设备、传输介质和构建小

型网络的方法。

5.4.1 网络的设备

网络互联的目的是使一个网络的用户能访问其他网络的资源,使不同网络上的用户能够互相通信和交换信息。由于各种网络使用的技术可能不同,所以要实现网络之间的互联则需要解决一些新问题。例如各种网络可能有不同的寻址方案、不同的最大分组长度、不同的超时控制、不同的差错恢复方法、不同的路由选择技术以及不同的用户访问控制等。另外各种网络提供的服务也可能不同,有的是面向连接的,有的是无连接的。网络互联技术就是要在不改变原来的网络体系结构的条件下,把一些异构型的网络互相连接成一个统一的通信系统,实现更大范围的资源共享。在网络互联时,一般不能简单地直接相连,而是通过一个中间设备来实现。按照 ISO/OSI 的分层原则,这个中间设备要实现不同网络之间的协议转换功能。根据它们工作的协议层不同进行分类,网络互联设备可以有:中继器(实现物理层协议转换,在电缆间转发二进制信号)、网桥(实现物理层和数据链路层协议转换)、路由器(实现网络层和以下各层协议转换)和网关(提供从最低层到传输层或以上各层的协议转换)。

1. 网络传输介质互联设备

网络线路与用户节点具体衔接时,需要使用网络传输介质的互联设备,如 T 型头(细同轴电缆连接器)、收发器、RJ-45 接口(屏蔽或非屏蔽双绞线连接器)、RS232 接口(目前微机与线路接口的常用方式)、DB-15 接口(连接网络接口卡的 AUI 接口)、VB35 同步接口(连接远程的高速同步接口)、网络接口单元和调制解调器(数字信号与模拟信号转换器)等。

2. 物理层的互联设备

物理层的互联设备有中继器(repeater)和集线器(hub)。

1) 中继器

它是在物理层上实现局域网网段互联的,用于扩展局域网网段的长度。由于中继器只在两个局域网网段间实现电气信号的恢复与整形,因此它仅适用于连接相同的局域网段。

从理论上说,可以用中继器把网络延长到任意长的传输距离,但是,局域网中接入的中继器的数量将受时延和衰耗的影响,因而必须加以限制。例如以太网中最多只能使用 4 个中继器。设计以太网连线时,两个最远用户之间的距离,包括用于局域网的连接电缆,不得超过 500m。即便使用了中继器,典型的 Ethernet 局域网应用要求从头到尾整个路径不能超过 1500m。中继器的主要优点是安装简便,使用方便,价格便宜。

2) 集线器

可以看成是一种特殊的多路中继器,亦具有信号放大功能。使用双绞线的以太网多

用 hub 扩大网络,同时也便于网络的维护。以集线器为中心的网络的优点是:当网络系统中某条线路或某节点出现故障时,不会影响网上其他节点的正常工作。集线器可分为无源(passive)集线器、有源(active)集线器和智能(intelligent)集线器。

无源集线器只负责把多段介质连接在一起,不对信号作任何处理,每一种介质段只允许扩展到最大有效距离的一半;有源集线器类似于无源集线器,但它具有对传输信号进行再生和放大从而扩展介质长度的功能;智能集线器除具有有源集线器的功能外,还可将网络的部分功能集成到集线器中,如网络管理,选择网络传输线路等。

3. 数据链路层的互联设备

数据链路层的互联设备有网桥和交换机。

1) 网桥(bridge)

网桥用于连接两个局域网段,工作于数据链路层。网桥要分析帧地址字段,以决定是否把收到的帧转发到另一个网段上。确切地说,网桥工作于 MAC 子层,只要两个网络 MAC 子层以上的协议相同,都可以用网桥互联。

网桥检查帧的源地址和目的地址,如果目的地址和源地址不在同一个网段上,就把帧转发到另一个网段上;若两个地址在同一个网段上,则不转发,所以网桥能起到过滤帧的作用。网桥的帧过滤特性很有用,当一个网络由于负载很重而性能下降时,可以用网桥把它分成两个网段并使得网段间的通信量保持最小。例如,把分布在两层楼上的网络分成每层一个网段,网段中间用网桥相连,这样的配置可以最大限度地缓解网络通信繁忙的程度,提高通信效率。同时由于网桥的隔离作用,一个网段上的故障不会影响到另一个网段,从而提高了网络的可靠性。

2) 交换机(switch)

交换机是一个具有简化、低价、高性能和高端口密集特点的交换产品,它是按每个包中的 MAC 地址相对简单地决策信息的转发。而这种转发决策一般不考虑包中隐藏的更深的其他信息。交换机转发数据的延迟很小,性能接近单个局域网,远远超过了普通桥接的转发性能。交换技术允许共享型和专用型的局域网段进行带宽调整,以减轻局域网之间信息流通的瓶颈问题。

交换机的工作过程为:当交换机从某一节点收到一个以太网帧后,将立即在其内存中的地址表(端口号—MAC 地址)进行查找,以确认该目的 MAC 的网卡连接在哪一个节点上,然后将该帧转发至该节点。如果在地址表中没有找到该 MAC 地址,也就是说,该目的 MAC 地址是首次出现,交换机就将数据包广播到所有节点。拥有该 MAC 地址的节点在接收到该广播帧后,将立即做出应答,从而使交换机将其节点的“MAC 地址”添加到 MAC 地址表中。

交换机的 3 种交换技术有端口交换(用于将以太网模块的端口在背板的多个网段之

间进行分配和平衡)、帧交换(其处理方式又分为:直通交换——提供线速处理能力,交换机只读出网络帧的前 14 个字节,便将网络帧传送到相应的端口上;存储转发——通过对网络帧的读取进行验错和控制;碎片丢弃——检查数据包的长度是否够 64 个字节。如果小于 64 字节,说明是假包,则丢弃该包,否则发送该包)和信元交换(采用长度固定的信元交换)。

虚拟网的划分是交换机的重要功能,实现的虚拟网的通常有 3 种形式。

(1) 静态端口分配:静态虚拟网的划分通常是由网管人员使用网管软件或直接设置交换机端口实现的,使其直接从属于某个虚拟网。这些端口一直保持这些从属性质,除非网管人员重新设置。此方法虽然比较麻烦,但比较安全,容易配置和维护。

(2) 动态虚拟网:支持动态虚拟网的端口,可以借助智能管理软件自动确定它们的从属。端口是通过网络包的 MAC 地址、逻辑地址或协议类型来确定虚拟网的从属性质的。当一网络节点刚接入网时,交换机端口还未分配,于是交换机通过读取网络节点的 MAC 地址动态地将该端口划入某个虚拟网。一旦网管人员这样配置好后,用户的计算机可以灵活地改变交换机端口,而不会改变该用户的虚拟网的从属性质,而且如果网络中出现了未定义的 MAC 地址,则可以向网管人员报警。

(3) 多虚拟网端口配置:该配置支持一用户或一端口可以同时访问多个虚拟网。可以将一台网络服务器配置成多个业务部门(每种业务设置成一个虚拟网)都可同时访问,(也可以同时访问多个虚拟网);还可让多个虚拟网间的连接只需一个路由端口即可完成,但这样会带来安全上的隐患。虚拟网的业界规范正在制定当中,因而各个公司的产品还谈不上互操作性。

4. 网络层互联设备

路由器(router)是网络层互联设备,用于连接多个逻辑上分开的网络。逻辑网络是指一个单独的网络或一个子网,当数据从一个子网传输到另一个子网时,可通过路由器来完成。

路由器具有很强的异种网互联能力,互联的网络最低两层的协议可以互不相同,通过驱动软件接口在第三层上得到统一。对于互联网络的第三层协议,如果相同,可使用单协议路由器进行互联;如果不同,则应使用多协议路由器。多协议路由器同时支持多种不同的网络层协议,并可以设置为允许或禁止某些特定的协议。所谓支持多种协议是指支持多种协议的路由,而不是指不同类型协议的相互转换。

通常把网络层地址信息叫做网络逻辑地址,把数据链路层地址信息叫做物理地址。路由器最主要的功能是选择路径。路由器的存储器中维护着一个路径表,记录各个网络的逻辑地址,用于识别其他网络。在互联网络中,当路由器收到从一个网络向另一个网络发送的信息包时,将剥离信息包的外层,解读信息包中的数据,获得目的网络的逻辑地址,

使用复杂的算法来决定信息经由哪条路径发送最合适,然后重新打包并转发出去。路由器的功能还包括过滤、存储转发、流量管理和介质转换等。一些功能较强的路由器还可有加密、数据压缩、优先和容错管理等功能。由于路由器工作于网络层,它处理的信息量比网桥要多,因而处理速度比网桥慢。

5. 应用层互联设备

网关(gateway)是应用层的互联设备。在一个计算机网络中,当连接不同类型而协议差别又较大的网络时,则要选用网关设备。网关的功能体现在 OSI 模型的最高层,它将协议进行转换,将数据重新分组,以便在两个不同类型的网络系统之间进行通信。由于协议转换是一件复杂的事,一般来说,网关只能进行一对一的转换,或是少数几种特定应用协议的转换,网关很难实现通用的协议转换。用于网关转换的应用协议有电子邮件、文件传输和远程工作站登录等。网关和多协议路由器(或特殊用途的通信服务器)组合在一起可以连接多种不同的系统。和网桥一样,网关可以是本地的,也可以是远程的。目前,网关已成为网络上每个用户访问大型主机的通用工具。

5.4.2 网络的传输介质

传输介质是信号传输的媒体,常用的介质分为有线介质和无线介质。有线介质有双绞线、同轴电缆和光纤等;无线介质有微波、红外线和微波等。

1. 有线介质

1) 双绞线(twisted-pair)

双绞线是现在最普通的传输介质,它由两条相互绝缘的铜线组成,典型直径为 1mm。双绞线分为屏蔽双绞线 STP 和非屏蔽双绞线 UTP,非屏蔽双绞线的线缆外皮为屏蔽层,适用于网络流量不大的场合。屏蔽式双绞线有一个金属甲套,对电磁干扰具有较强的抵抗能力,适用于网络流量较大的高速网络协议应用。双绞线又可分为 3 类、4 类、5 类、6 类和 7 类双绞线。现在常用的是 5 类 UTP,其频率带宽为 100MHz。6 类和 7 类双绞线可分别工作于 200MHz 和 600MHz 的频率带宽之上,且采用特殊设计的 RJ45 插头。

双绞线大多应用于 10BASE-T 和 100BASE-T 的以太网中,具体规定有:一段双绞线的最大长度为 100m,只能连接一台计算机;双绞线的每端需要一个 RJ45 插头;各段双绞线通过集线器相连,利用双绞线最多可连接 64 个站点到中继器。

2) 同轴电缆(coaxial)

同轴电缆也像双绞线那样由一对导体组成。同轴电缆又分为基带同轴电缆(阻抗为 50 Ω)和宽带同轴电缆(阻抗为 75 Ω)。基带同轴电缆用来直接传输数字信号,它又分为粗同轴电缆和细同轴电缆。其中粗同轴电缆适用于较大局域网的网络干线,布线距离较长,可靠性较好,但是网络安装和维护等方面比较困难,造价较高。而细同轴电缆安装较容

易,而且造价较低,但因受网络布线结构的限制,其日常维护不甚方便。宽带同轴电缆用于频分多路复用(FDM)的模拟信号发送,还用于不使用频分多路复用的高速数字信号发送和模拟信号发送。闭路电视所使用的 CATV 电缆就是宽带同轴电缆。

3) 光导纤维(fiber optic)

光导纤维简称光纤,它重量轻,体积小。用光纤来传输电信号时,发送端先要将其转换成光信号,而接收端则要用光检波器把光信号还原成电信号。光纤是软而细的、利用内部全反射原理来传导光束的传输介质。按光源采用的发光管(发光二极管和注入型激光二极管)不同,光纤可以分为多模光纤(multimode fiber)和单模光纤(single mode fiber)。多模光纤使用的材料是发光二极管,价格较便宜,但定向性较差。单模光纤使用的材料是注入型二极管,定向性好,损耗少,效率高,传播距离长,但价格昂贵。

2. 无线介质

无线传输介质不需要架设或铺埋电缆或光纤,而是通过大气传输。目前有微波、红外线和激光 3 种技术。

1) 微波

微波通信是在对流层视线距离范围内利用无线电波进行传输的一种通信方式,频率范围为 2~40GHz。微波通信是沿直线传播的,由于地球表面是曲面,微波在地面上的传播距离有限。直接传播的距离与天线的高度有关,天线越高距离越远,但超过一定距离后就要用中继站来接力。两微波站的通信距离一般为 30~50km,长途通信时必须建立多个中继站。中继站的功能是变频和放大,进行功率补偿。

微波通信分为模拟微波通信和数字微波通信两种。模拟微波通信主要采用调频制,数字微波通信大都采用相移键控(PSK)技术。微波通信的传输质量比较稳定,影响质量的主要因素是雨雪天气对微波产生的吸收损耗,不利地形或环境对微波所造成的衰减现象。

2) 红外线和激光

红外通信和激光通信也像微波通信一样,有很强的方向性,都是沿直线传播的。这 3 种技术都需要在发送方和接收方之间有一条视线(Line-of-sight)通路,有时统称这三者为视线媒体。所不同的是,红外通信和激光通信把要传输的信号分别转换为红外光信号和激光信号,直接在空间传播。

微波、红外和激光这 3 种视线媒体不需要铺设电缆,但对环境气候较为敏感,例如雨、雾和雷电。相对来说,微波一般对雨和雾的敏感度较低。

3) 卫星通信

卫星通信是以人造卫星为微波中继站,它是微波通信的特殊形式。卫星接收来自地面发送站发出的电磁波信号后,再以广播方式用不同的频率发回地面,为地面工作站接收。卫

星通信可以克服地面微波通信距离的限制。一个同步卫星可以覆盖地球的三分之一以上的表面,3个同步卫星就可以覆盖地球上的全部通信区域,这样地球上的各个地面站之间都可互相通信了。由于卫星信道频带宽,可采用频分多路复用技术将其分为若干个子信道,有些用于地面站向卫星发送(称为上行信道),有些用于卫星向地面转发(称为下行信道)。卫星通信的优点是容量大、距离远,缺点是传播延迟时间长。

5.4.3 网络的构建

在一个局域网里,其基本组成部件为服务器、客户机、网络设备、通信介质和网络软件等。

(1) 服务器(server): 局域网的核心,根据它在网络中的作用,可进一步分为文件服务器、打印服务器和通信服务器。

(2) 客户机(client): 客户机又称为用户工作站,是用户与网络应用的接口设备。每一个客户机既要运行本机的进程又要和服务器打交道,同时,还可能接受来自其他工作站的信息,在网中和其他工作站一起构成了可进行分布式处理的环境。它还可分享网上的资源,并可为其他工作站提供必要的支持。

(3) 网络设备: 主要指一些硬件设备,如网卡、收发器、中继器、集线器、网桥和路由器等。网卡是一种必不可少的网络设备。常用的网卡有 Ethernet(以太网)网卡、ARCnet 网卡、ESIA 总线网网卡和 Token-Ring 网卡等。

(4) 通信介质: 数据的传输媒体。不同的通信介质有着不同的传输特性。

(5) 网络软件: 网络软件主要包括底层协议软件和网络操作系统(NOS)等。底层协议软件由一组标准规则及软件构成,使实体间或网络之间能够互相进行通信。网络操作系统主要管理整个网络的资源和运行,并为用户提供应用接口。

在广域网中,除了以上的设备外,还应该还有通信处理机、集线器、信号变换器(调制解调器)和多路复用器等。通信处理机也称通信控制处理机或前端处理机,是连接主计算机与通信线路单元的计算机,负责通信控制和通信处理工作。集线器的作用是把若干个终端用低速线路先集中起来,连接到高速线路上,经高速线路再与通信处理机连接,用以提高通信效率,减少通信费用。信号变换器提供不同信号之间的变换,不同传输介质采用不同类型的信号变换器。当用电话线作为传输线时,电话线只能传输模拟信号,但主计算机和终端输出的是数字信号,因此在通信线路与主计算机、通信处理机和终端之间均需接入转换模拟信号与数字信号的变换器。

【例 5.1】 为了将两个相邻办公室的多台计算机连接成一个局域网,以方便传输文件,共享资源,最简单的连接方式如图 5-11 所示。

采用集线器(hub)将两个办公室的多台计算机连接成一个局域网。如果一个 hub 的

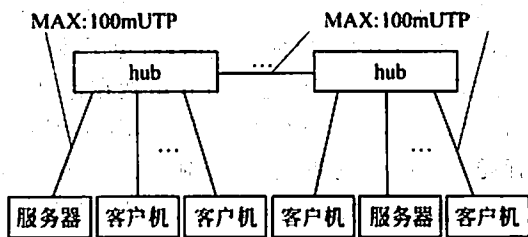


图 5-11 简单网络连接方式

端口不足,可采用多个 hub 进行级联,构成总线或星形拓扑结构。传输介质采用 5 类以上的非屏蔽双绞线 UTP,使用 RJ45 连接 hub 与计算机。网卡采用 10/100M 自适应的以太网卡。协议使用 TCP/IP、NetBEUI 或其他协议。该网络的安装和维护简单易行且费用低廉,计算机和 hub 之间的最大 UTP 电缆长度为 100m,两个计算机之间(即端一端)最多允许有 4 个 hub 和 5 个电缆段,即最大网络长度为 500m。

【例 5.2】 某公司为了在单位内部构建一个局域网,以便能在网上使用物资管理系统、人事管理系统以及财务管理系统等,实现电子文件的上传下达等事宜,要求网络的速度较高。最简单、经济的连接方式如图 5-12 所示。

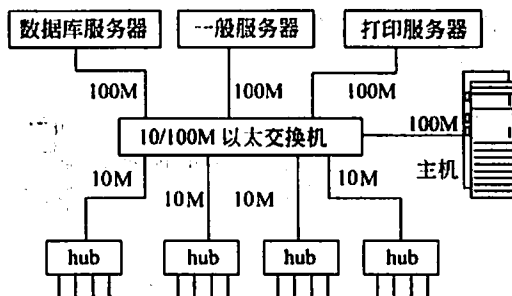


图 5-12 某公司局域网连接方式

采用一台 10/100M 以太网交换机和多个 hub 来构建单位内部的局域网。10/100M 以太网交换机与各种服务器、主机之间的数据传输速度为 100M,与其他 hub 之间的数据传输速度为 10M/100M。hub 再与多个客户机(共享 10M/100M 带宽)相连,构成星形拓扑结构。传输介质采用 5 类以上的非屏蔽双绞线 UTP,使用 RJ45 连接 hub 与计算机。网卡采用 10/100M 自适应的以太网卡。协议使用 TCP/IP 或 NetBEUI 或其他协议。

【例 5.3】 某家庭想利用 ADSL 连接 Internet,基于以下原因:

(1) ADSL 具有很高的传输速率(下行 2~8Mb/s,上行 64Kb/s~640Kb/s),为普通拨号 modem 的百倍以上,也是宽带上网中速度较高的一种;

(2) ADSL 上网和打电话互不干扰。ADSL 数据信号和电话音频信号采用频分复用原理,各自频段互不干扰,上网的同时可以使用电话,避免了拨号上网的烦恼;

(3) ADSL 独享带宽,安全可靠。其他宽带方式虽然在速度方面有快过它的,但有的是属于共享带宽方式,如 cable modem,下行可达到 20Mb/s,但它是一种粗糙的总线型广播网络,成千上万用户争抢 20Mb/s 的带宽,而非独享。更为严重的是它属于总线型网络,先天的广播特性,造成了信息传输的不安全性。ADSL 利用中国电信深入千家万户的电话网络,已经形成星状的网络拓扑结构。骨干网络采用中国电信遍布全城全国的光纤传输,独享 2~8Mb/s 带宽,信息传递速度快,安全可靠;

(4) ADSL 费用低廉,虽然电话线同时传递电话语音和数据,但数据并不通过电话交换机,因此不用拨号,一直在线,属于专线上网方式。这意味着使用 ADSL 上网不需要缴纳拨号上网的电话费用。另一方面不需要对原有电话线路进行改造,用户端不需要购买价格昂贵的设备,只需一个现在相当普及的 ADSL modem 即可,而不像 FTTB+LAN 光纤到楼宽带方式一样需花费一大笔资金来进行线路改造,相对来说投资较少;

(5) ADSL 能提供真正的视频点播 VOD、网上游戏、交互电视以及网上购物等宽带多媒体服务,远程 LAN 接入、远地办公室以及在家工作等高速数据应用,远程医疗、远程教育、远地可视会议和体育比赛现场实时传送等服务。

因此决定申请一条 ADSL 线路,通过拨号来连接 Internet。

使用 ADSL 上网所需的硬件设备有一块网卡和 ADSL modem。当然家用电脑也是必须的。连接方式如图 5-13 所示。

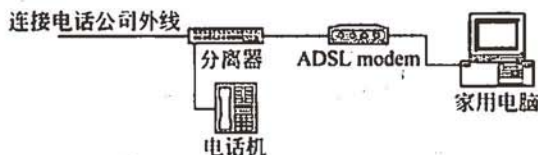


图 5-13 ADSL 单用户接入方式

如果不是家庭,而是某公司需要共享 ADSL 来连接 Internet,则接入方式如图 5-14 所示。

需要注意的问题:

- 接口方式 有以太网、USB 和 PCI 3 种。USB 和 PCI 适用于家庭用户,性价比高,小巧,方便实用;外置以太网口的产品只适用于企业和办公室的局域网,它可以带多台机器进行上网。有的以太网接口的 ADSL modem 同时具有桥接和路由功能。
- 分离器 购买 ADSL modem 时,一定要检查是否带有分离器。若没有附带,则需单独购买。

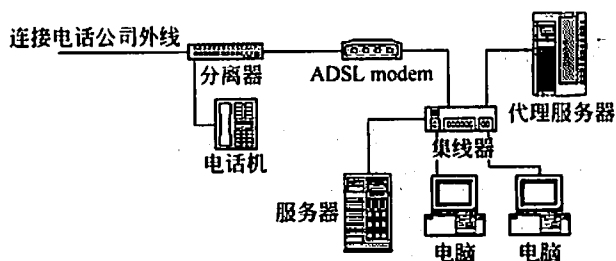


图 5-14 ADSL 共享接入方式

- 支持的协议 ADSL modem 上网拨号方式有专线方式(静态 IP)、PPPoA 和 PPPoE 3 种。普通用户多采用 PPPoE(Point-to-Point Protocol over Ethernet)或 PPPoA(Point-to-Point Protocol over ATM)虚拟拨号的方式来上网。

【例 5.4】某校园/大型企业为本单位建设高速信息网络,网络主干中心为千兆以太网的光纤局域网,连接各学院、系和图书馆等信息网,并接入 Internet。实现各级各类网络的互联互通,为学校的各级单位、教师和学生提供方便、快捷的信息与教学服务,从而有效地为科研和教学服务,提高学校的整体水平。网络的构建如图 5-15 所示。

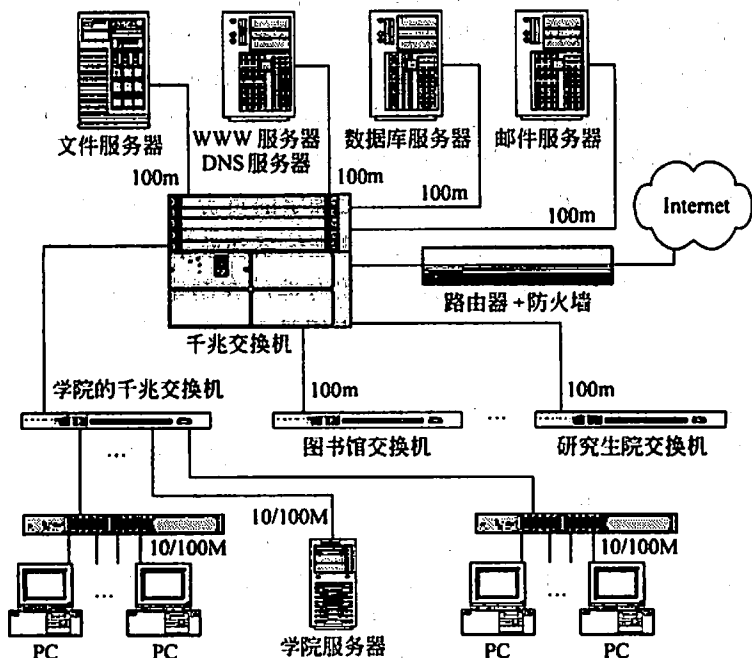


图 5-15 校园网的构建

5.5 Internet 及应用

Internet 是世界上规模最大,覆盖面最广且最具影响力的计算机互联网络,它将分布在世界各地的计算机利用开放系统协议连接在一起,用来进行数据传输、信息交换和资源共享。在现阶段,Internet 作为未来信息高速公路的雏形,无论在科学研究、教育和金融,还是在商业和军事等部门,其影响都越来越大。

5.5.1 Internet 概述

从用户的角度来看,整个互联网在逻辑上是统一的、独立的,在物理上则由不同的网络互联而成。从技术角度看,Internet 本身不是某一种具体的物理网络技术,它是能够互相传递信息的众多网络的一个统称,或者说它是一个网间网。只要人们进入了这个互联网,就是在使用 Internet。正是由于 Internet 的这种特性,使得广大 Internet 用户不必关心网络的连接,而只关心网络提供的丰富资源。

连入 Internet 的计算机网络种类繁多,形式各异,且分布在世界各地。因此,需要通过路由器(IP 网关)并借助各种通信线路或公共通信网络把它们连接起来。由于实现了与公用电话网的互联,个人用户入网十分方便,只要有电话和 modem 即可,这也是 Internet 迅速普及的原因之一。Internet 由美国的 ARPAnet 网络发展而来,因此,它沿用了 ARPAnet 使用的 TCP/IP 协议。由于该协议非常有效且使用方便,许多操作系统都支持它,无论是服务器还是个人计算机都可安装使用。

对于全球性最大的互联网络,总的说来并无确定的负责人,它是由各自独立管理的网络互联构成的,而这些网络都各自拥有自己的管理体系和政策法规。因此,没有集中的负责掌管整个 Internet 的机构。尽管如此,某些政府部门在制定 Internet 有关政策时实际上仍起着主导作用。Internet 目前的最高国际组织是 Internet 学会(Internet Society),该学会是一个志愿者组织,也是一个非盈利性的专业化组织,其主要目标是促进 Internet 的改革与发展。该学会下分 Internet 体系结构研究会(IAB)和其他几个研究会。IAB 下又有工程组(IETF)、许可证管理局(ICRS)、技术研究组(IRTF)和编号管理局(IANA)等。IAB 的主要任务是为 Internet 的科研与开发提供支持与服务。

在 Internet 中,分布着一些覆盖范围很广的大网络,这种网络称为“Internet 主干网”,它们一般属于国家级的广域网。例如,CHINANET 和 CERNET 等就是中国的 Internet 主干网。主干网一般只延伸到一些大城市或重要地方,在那里设立主干网节点。每一个主干网节点可以通过路由器将广域网与局域网连接起来,一个节点还可以通过另外的路由器再与其他局域网互联,由此形成一种网状结构。

5.5.2 Internet 地址

无论在网检索信息还是发送电子邮件,都必须知道对方的 Internet 地址。它能惟一地确定 Internet 上每一台计算机或每个用户的位置。也就是说,Internet 上每一台计算机或每个用户都有惟一的地址来标识它的身份和位置,以便于区分几千万个用户、几百万台计算机和成千上万的组织。Internet 地址格式主要有两种书写形式:域名格式和 IP 地址格式。

1. 域名

域名(domain name)通常是用户所在的主机名字或地址。域名格式由若干部分组成,每个部分又称子域名,它们之间用“.”分开,每个部分最少由两个字母或数字组成。域名通常按分层结构来构造,每个子域名都有其特定的含义。通常情况下,一个完整、通用的层次型主机域名由 4 部分组成:

计算机主机名.本地名.组名.最高层域名

从右到左,子域名分别表示不同的国家或地区的名称(只有美国可以省略表示国家的顶级域名)或组织类型、组织名称、分组织名称和计算机名称等。域名地址的最后一部分子域名称为高层域名(或顶级域名),它大致上可以分成两类:一类是组织性顶级域名;另一类是地理性顶级域名。

例如:www.dzkjdx.edu.cn 中的“cn”是地理性顶级域名,表示“中国”。

www.263.net 中的“net”是组织性顶级域名,表示“网络技术组织机构”。

如果一个主机所在的网络级别较高,它拥有的域名可能仅有 3 部分:本地名.组名.最高层域名。现在,Internet 地址管理机构 IPRA(Internet PCA Registration Authority)和 IANA(Internet Assigned Number Authority)负责 Internet 最高层域名的登记和管理。

2. IP 地址

Internet 地址是按名字来描述的,这种地址表示方式易于理解和记忆。但 Internet 中的主机地址是用 IP 地址来惟一标识的,这是因为 Internet 中所使用的网络协议是 TCP/IP 协议,故每个主机必须用 IP 地址来标识。

每个 IP 地址都由 4 个小于 256 的数字组成,数字之间用“.”分开。Internet 的 IP 地址共有 32 位,4 个字节。它有两种表示格式:二进制格式和十进制格式。二进制格式是计算机所认识的格式,十进制格式是由二进制格式“翻译”过去的,主要是为了便于使用和掌握。例如,十进制 IP 地址 129.102.4.11 的表示方法与二进制的表示方法 10000001 01100110 00000100 00001011 相同,显然表示成带点的十进制格式要方便得多。

域名和 IP 地址是一一对应的,域名易于记忆,便于使用,因此得到比较普遍的使用。

当用户和 Internet 上的某台计算机交换信息时,只需要使用域名,网络则会自动地将其转换成 IP 地址,找到该台计算机。

Internet 中的地址可分为 5 类: A 类、B 类、C 类、D 类和 E 类。各类地址的分配方案如图 5-16 所示。在 IP 地址中,全 0 代表网络,全 1 代表广播。

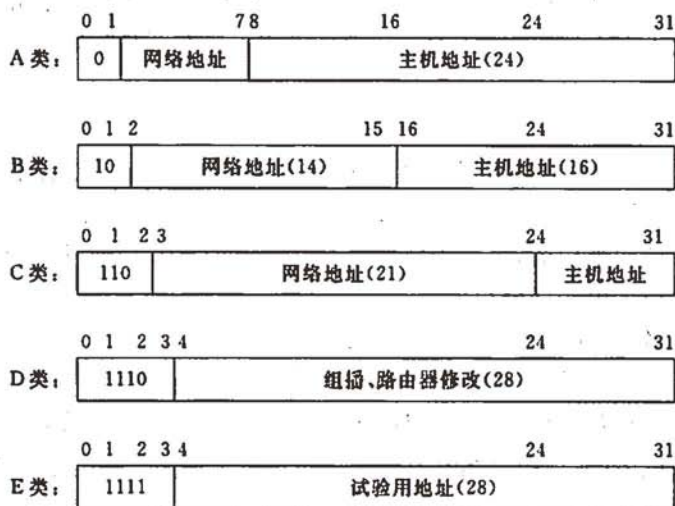


图 5-16 各类地址分配方案

A 类网络地址占有 1 字节(8 位),最高位为 0 表示 A 类网络地址,余下 7 位为真正的网络地址,支持 1~126 个网络。后面的 3 字节(24 位)为主机地址,共提供 $2^{24}-2$ 个主机地址。A 类网络地址第一个字节的十进制值范围为: 000~127。

B 类网络地址占 2 字节,最高两位为“10”表示 B 类网络地址,其余 14 位为真正的网络地址。主机地址占后面的两个字节(16 位),所以 B 类全部主机地址有: $(2^{14}-2) \times (2^{16}-2) = 16382 \times 65534$ 个。B 类网络地址第一个字节的十进制值范围为: 128~191。

C 类网络地址占有 3 字节,它是最通用的 Internet 地址。最高 3 位为“110”表示 C 类网络地址,其余 21 位为真正的网络地址,因此 C 类地址支持 $2^{21}-2$ 个网络。主机地址占最后 1 字节,每个网络可多达 2^8-2 个主机。C 类网络地址第一个字节的十进制值范围为: 192~223。

D 类地址是相当新的。它的识别头是 1110,用于组播,例如用于路由器修改。D 类网络地址第一个字节的十进制值范围为: 224~239。

E 类地址为保留地址,其识别头是 1111。E 类网络地址第一个字节的十进制值范围为: 240~255。

网络软件和路由器使用子网掩码(subnet mask)来识别报文是仅存放在网络内部还是被路由转发到其他地方。在一个字段内“1”的出现表明一个字段包含所有或部分网络地址。“0”表明主机地址位置。例如,最常用的 C 类地址使用前 3 个 8 位来识别网络,最后一个 8 位识别主机。因此,子网掩码是 255.255.255.0。

子网地址掩码是相对于 IP 地址而言的,如果脱离了 IP 地址就毫无意义。它的出现一般是跟着一个特定的 IP 地址,用来为计算这个 IP 地址中的网络号部分和主机号部分提供依据。换句话说,就是在写一个 IP 地址后,再指明哪些是网络号部分,哪些是主机号部分。子网掩码的格式与 IP 地址相同,所有对应网络号的部分用 1 填上,所有对应的主机号部分用 0 填上。A 类、B 类和 C 类 IP 地址类默认的子网掩码如表 5-2 所示。

表 5-2 带点十进制符号表示的默认子网掩码

地址类	子网掩码位	子网掩码
A 类	11111111 00000000 00000000 00000000	255.0.0.0
B 类	11111111 11111111 00000000 00000000	255.255.0.0
C 类	11111111 11111111 11111111 00000000	255.255.255.0

如果需要将网络进行子网划分,此时子网掩码可能不同于以上默认的子网掩码。例如,138.96.58.0 是一个 8 位子网化的 B 类网络 ID。基于 B 类的主机 ID 的前 8 位被用来表示子网化的网络,对于网络 138.96.39.0,其子网掩码应为 255.255.255.0。

例如:一个 B 类地址 172.16.3.4,为了直观地告诉大家前 16 位是网络号,后 16 位是主机号,就可以附上子网掩码:255.255.0.0(11111111 11111111 00000000 00000000)。

假定某单位申请的 B 类地址为 179.143.XXX.XXX。如果希望把它划分为 14(至少占二进制的 4 位)个虚拟的网络,则需要占 4 位主机地址位,使用子网掩码 255.255.240.0~255.255.255.0 来建立子网。每个 LAN 可有 $2^{12}-2$ 个主机,且各子网可具有相同的主机地址。

假设一个组织有几个相对大的子网,每个子网包括 25 台左右的计算机。同时又有一些相对较小的子网,每个子网大概只有几台计算机。在这种情况下,可以将一个 C 类地址分成 6 个子网(每个子网可以包含 30 台计算机),这样就可以解决问题。但这里有一个新的情况,那就是大的子网基本上完全利用了 IP 地址范围,但是小的子网却造成了许多 IP 地址的浪费。为了解决这个新的难题,避免任何 IP 浪费,就出现了采用不同大小的子网掩码对 IP 地址空间进行子网划分的解决方案,这种新的解决方案叫作可变长子网掩码 VLSM。

VLSM 采用一种十分直观的表达方法,那就是在 IP 地址后面加上“/网络号及子网络号编址比特数”。例如 193.168.125.0/27,就表示前 27 位为网络号。

例如,给定 135.41.0.0/16 的网络 ID,所需的配置是为将来使用保留一半的地址,其余的生成 15 个子网,达到 2000 台主机。

由于要为将来使用保留一半的地址,完成了 135.41.0.0 的网络 ID 的 1 位子网化,生成 135.41.0.0/17 和 135.41.128.0/17 两个子网,子网 135.41.128.0/17 被选作为将来使用所保留的地址部分,135.41.0.0/17 被继续生成子网。

为达到划分 2000 台主机,15 个子网的要求,需要将 135.41.128.0/17 的子网化的网络 ID 的 4 位子网化。这就产生了 16 个子网(135.41.128.0/21,135.41.136.0/21,...,135.41.240.0/21,135.41.248.0/21),允许每个子网有 2046 台主机。最初的 15 个子网化的网络 IDs(135.41.128.0/21 到 135.41.240.0/21)被选定为网络 IDs,从而实现了要求。

现在的 IP 协议的版本号为 4,所以也称之为 IPv4,为了方便网络管理员阅读和理解,使用了 4 个十进制数中间加小数点“.”来表示。但随着互联网的发展,IPv4 不论从地址空间上,还是协议的可用性上都无法满足互联网的新要求。这样出现一个新的 IP 协议 IPv6,使用 8 个十六进制数中间加小数点来表示。IPv6 将原来的 32 位地址扩展成为 128 位地址,彻底解决了地址缺乏的问题。

5.5.3 Internet 服务

作为全世界最大的国际性计算机网络的 Internet,为全球的科研界、教育界和娱乐界等方面提供了极其丰富的信息资源和最先进的信息交流手段。在 Internet 上,时刻传送着大量的各种各样的信息,从电影、实况转播到最尖端的科学研究等无所不包,当然信息最多的还是科技信息,如计算机软件、科技论文、图书馆/出版社目录、最新科技动态、电子杂志、产品推销以及网络新闻等。而这些内容均可由 Internet 服务来为用户提供。

使用传输控制协议或用户数据报协议时,Internet IP 可支持 65 535 种服务,这些服务是通过各个端口到名字实现的逻辑连接。端口分两类:一类是已知端口或称公共端口,端口号从 0 到 1023,这些端口由 Internet 分配地址和端口号的组织 IANA(Internet Assigned Number Authority)分配;另一类须在 IANA 注册登记的端口号,从 1024 到 65 535。

前面介绍了 Internet 网络接口层、网际层协议和传输层协议,本节主要介绍 Internet 的高层协议,如 DNS 域名服务、TELNET 远程登录服务、E-mail 电子邮件服务、WWW 服务和 FTP 文件传输服务等。

1. DNS 域名服务

Internet 中的域名地址与 IP 地址是对应的,它们之间是通过域名服务 DNS 来完成映射变换的。实际上,DNS 是一种分布式地址信息数据库系统,服务器中通常仅包含整

个数据库的一部分信息,并供客户查询。DNS 允许局部控制整个数据库的某些部分,但数据库的每一部分都可通过全网查询得到。

域名系统采用的是客户机/服务器模式,整个系统由解析器和域名服务器组成。解析器是客户方,它负责查询域名服务器,解释从服务器返回来的应答信息,将信息返回给请求方等工作。域名服务器(domain name server)是服务器方,它通常保存着一部分域名空间的全部信息,这部分域名空间称为区(zone)。一个域名服务器可以管理一个或多个区。域名服务器可以分为主服务器、Caching Only 服务器和转发服务器(forwarding server)。

域名系统是一个分布式系统,其管理和控制也是分布式的。当用户 A 在查找另一用户 B 时,域名系统的工作过程如下:

- ① 解析器向本地域名服务器发出查阅用户 B 域名的请求。
- ② 本地域名服务器向最高层域名服务器发出查询地址的请求。
- ③ 最高层域名服务器返回给本地域名服务器一个 IP 地址。
- ④ 本地域名服务器向组域名服务器发出查询地址的请求。
- ⑤ 组域名服务器返回给本地域名服务器一个 IP 地址。
- ⑥ 本地服务器向刚返回的域名服务器发出查询域名地址的请求。
- ⑦ 然后 IP 地址返回给本地域名服务器。
- ⑧ 本地域名服务器将该地址返回给解析器。

因此,为了得到一个 IP 地址,本地域名服务器常常需要查询多个域名服务器。在查询地址的同时,本地域名服务器可以得到许多其他域名服务器的信息,像它们的 IP 地址、所负责的区域等。本地域名服务器将这些信息连同最终查询到的主机 IP 地址全部存放在它的 Cache 中,以便将来参考。当下次解析器再查询与这些域名相关的信息时,就可以直接引用。这样,就大大减少了查询时间。

因此,访问主机的时候只需要知道域名,通过 DNS 服务器再将域名变换为 IP 地址。DNS 所用的是 UDP 端口,端口号为 53。

2. TELNET 远程登录服务

远程登录服务是在 Telnet 协议的支持下,将用户计算机与远程主机连接起来,将本地的输入送给远程计算机,在远程计算机上运行程序,最终将相应的输出传送到本地机器上显示。由于这种服务基于 Telnet 协议且使用 Telnet 命令进行远程登录,故称为 Telnet 远程登录。

Telnet 是基于客户机/服务器模式的服务系统,它由客户软件、服务器软件以及 Telnet 通信协议等 3 部分组成。远程计算机又称为 Telnet 主机或服务器,本地计算机作为 Telnet 客户机来使用,起到远程主机的一台虚拟终端(仿真终端)的作用。通过它用户可以同主机上的其他用户一样共同使用该主机提供的服务和资源。

当用户使用 Telnet 登录远程主机时,该用户必须在这个远程主机上拥有合法的账号和相应的密码,否则远程主机将会拒绝登录。在运行 Telnet 客户程序时,首先应该建立与远程主机的 TCP 连接。从技术上讲,就是在一个特定的 TCP 端口(端口号一般为 23)上打开一个套接字。如果远程主机上的服务器软件一直在这个众所周知的端口上侦听连接请求,则这个连接便会建立起来。此时用户的计算机就会成为该远程主机的一个终端,便可以进行联机操作了。即以终端方式为用户提供人机界面,然后将用户输入的信息通过 Telnet 协议便可以传送给远程主机。主机侦听到用户的请求并处理后,将处理的结果通过 Telnet 协议返回给客户程序。客户机收到远程主机发送来的信息,经过适当的转换,最后显示在用户计算机的屏幕上。

3. E-mail 电子邮件服务

电子邮件(E-mail)就是利用计算机进行信息交换的电子媒体信件。它是随着计算机网络出现的,并依靠网络的通信手段实现普通邮件信息的传输,是应用最广泛的一种服务。

电子邮件通过计算机网络与其他用户进行联系是一种快速、简便、高效和廉价的现代化通信手段。如果想使用 E-mail,首先必须拥有一个电子邮箱。电子邮箱是由 E-mail 服务提供者在 E-mail 服务器磁盘上为用户建立的专用于存放电子邮件的存储区域,并由 E-mail 服务器进行管理。用户可使用 E-mail 客户软件在自己的电子邮箱里来收发电子邮件。电子邮件地址的一般格式为:用户名@主机名。例如 fqzhang@china.com。

E-mail 系统基于客户机/服务器模式,整个系统由 E-mail 客户软件、E-mail 服务器和通信协议等 3 部分组成。E-mail 客户软件也称用户代理(User Agent),是用户用来收发和管理电子邮件的工具。E-mail 服务器主要充当“邮局”的角色,它除了为用户提供电子邮箱外,还承担着信件的投递业务。当用户发送一个电子邮件后,E-mail 服务器通过若干网络中间节点“存储-转发”式的传递,最终把信件投递到目的地(收信人的电子邮箱);E-mail 服务器主要采用 SMTP 协议,本协议描述了电子邮件的信息格式及其传递处理方法,保证被传送的电子邮件能够正确地寻址和可靠地传输。它是面向文本的网络协议,其缺点是不能用来传送非 ASCII 码文本和非文字性附件,在日益发展的多媒体环境中以及人们关注的邮件私密性方面,更显出它的局限性。后来的一些协议,包括多用途的 Internet 邮件扩充协议 MIME(Multi-Purpose Internet Mail Extensions)及增强私密邮件保护协议 PEM(Privacy Enhanced Mail),弥补了 SMTP 协议的不足。SMTP 协议通常用在大型多用户、多任务的操作系统环境中,将它用在 PC 机上收信是十分困难的。所以,TCP/IP 网络上的大多数邮件管理程序均使用 SMTP 协议发信,采用 POP(Post Office Protocol)协议(常用的是 POP3)保管用户未能及时取走的邮件。

POP 协议有两个版本:POP2 和 POP3。目前使用的 POP3 既能与 STMP 共同使

用,也可以单独使用,以传送和接受电子邮件。POP 协议是一种简单的纯文本协议,每次传输均以整个 E-Mail 为单位,不能提供部分传输。

如果用户要传送 E-mail,首先需在联网的计算机上使用邮件软件编好邮件正文,填好邮件收信人的 E-mail 地址、发信人电子邮件地址(或自动填上)以及邮件的主题等内容,然后使用 E-mail 的发送命令发出。此时,E-mail 发送端与接收端的计算机并不直接进行通信。发信端计算机送出邮件后,邮件先到达自己所注册的邮件服务器主机,再在网络传输过程中经过多个计算机和路由器的中转,到达目的地的邮件服务器主机,送进收信人的电子邮箱。当邮件的接收者上网并启动电子邮件管理程序,它就会自动检查邮件服务器中的电子邮箱,一旦发现新邮件,就可立刻下载到自己的计算机上,完成接收邮件的任务。

收发电子邮件的软件工具有很多,许多浏览器如 Internet Explore 和 Netscape 也带有邮件管理功能。目前比较受欢迎的是 IE 6.0 的组件 Outlook Express。

简单邮件传送协议 SMTP 和用于接收邮件的 POP3 协议均利用 TCP 端口。SMTP 所用的端口号是 25,POP3 所用的端口号是 110。

4. WWW(World Wide Web)服务

万维网 WWW 是一种交互式图形界面的 Internet 服务,具有强大的信息连接功能,是目前 Internet 中最受欢迎的、增长速度最快的一种多媒体信息服务系统。

万维网是基于客户机/服务器模式的信息发送技术和超文本技术的综合,WWW 服务器把信息组织为分布的超文本,这些信息节点可以是文本、子目录或信息指针。WWW 浏览程序为用户提供基于超文本传输协议 HTTP(hyper text transfer protocol)的用户界面,WWW 服务器的数据文件由超文本标记语言 HTML(hyper text markup language)描述。HTML 利用统一资源定位地址 URL 指向超媒体链接,并在文本内指向其他网络资源。

超文本传输协议(HTTP)是一个 Internet 上的应用层协议,是 Web 服务器和 Web 浏览器之间进行通信的语言。所有的 Web 服务器和 Web 浏览器必须遵循这一协议,才能发送或接收超文本文件。HTTP 是客户机/服务器体系结构,提供信息资源的 Web 节点(即 Web 服务器)可称作 HTTP 服务器,Web 浏览器则是 HTTP 服务器的客户端。WWW 上的信息检索服务系统就是遵循 HTTP 协议运行的。在 HTTP 的帮助下,用户可以只关心要检索的信息,而无需考虑这些信息存储在什么地方。

在 Internet 上,万维网整个系统由 Web 服务器、浏览器(Browser)和 HTTP 通信协议等 3 部分组成。Web 服务器提供信息资源,Web 浏览器将信息显示出来,HTTP 是为分布式超媒体信息系统而设计的一种网络协议,主要用于域名服务器和分布式对象管理。它能够传送任意类型数据对象,以满足 Web 服务器与客户之间多媒体通信的需要,从而成为 Internet 中发布多媒体信息的主要协议。

统一资源定位器 URL 是在 WWW 中标识某一特定信息资源所在位置的字符串,是一个具有指针作用的地址标准。在 WWW 上查询信息,必不可少的一项操作是在浏览器中输入查询目标的地址,这个地址就是 URL,也称 Web 地址,俗称“网址”。一个 URL 指定一个远程服务器域名和一个 Web 页。换言之,每个 Web 页都有惟一的 URL。URL 也可指向 FTP、WAIS 和 gopher 服务器代表的信息。通常,用户只需要了解和使用主页的 URL,通过主页再访问其他页。当用户通过 URL 向 WWW 提出访问某种信息资源时,WWW 的客户服务器程序自动查找资源所在的服务器地址,一旦找到,立即将资源调出供用户浏览。

使用 WWW 的浏览程序(例如 Internet Explore、Netscape 或 Mosaic 等),网页的超文本链接将引导用户找到所需要的信息资源。超文本文档包含一些标题和章节等构造文本的命令,从而允许浏览程序格式化为一种文本类型,以获得最佳的屏幕显示效果。有的浏览程序还可调用其他应用程序,以显示特殊类型的文档。

如果你已经是 Internet 的用户,只要在自己的计算机上运行一个客户程序(WWW 浏览器),并给出需访问的 URL 地址,就可以尽情浏览这些来自远方或近邻的各种信息。WWW 工作过程为:首先通过局域网,或通过电话拨号连入 Internet,并在本地计算机上运行 WWW 浏览器程序。然后根据想要获得的信息来源,在浏览器的指定位置输入 WWW 地址,并通过浏览器向 Internet 发出请求信息。此时,网络中的 IP 路由器和服务器将按照地址把信息传递到所要求的 WWW 服务器中。而 WWW 服务器则不断地在一个周知的 TCP 端口(端口号为 80)上侦听用户的连接请求,当服务器收到请求后,找到所要求的 WWW 页面。最后,服务器将把找到的页面通过 Internet 传送回用户的计算机。浏览器接收传来的超文本文件,转换并显示在计算机屏幕上。

一个 URL(Web 地址)包括以下几部分:协议、主机域名、端口号(任选)、目录路径(任选)和一个文件名(任选)。其格式为:

scheme://host, Domain[: port]Upath/filename]

其中,scheme 指定服务连接方式(协议)。通常有下列几种:

- file: 本地计算机上的文件。
- ftp: FTP 服务器上的文件。
- gopher: Gopher 服务器上的文件。
- http: WWW 服务器上的超文本文件。
- New: 一个 USenet 的新闻组。
- telnet: 一个 Telnet 站点。
- wais: 一个 WAIS 服务器。

- mailto: 发送邮件给某人。

在地址的冒号之后通常是两个反斜线,表示后面是指定信息资源的位置。其后是一个可选的端口号,地址的最后部分是路径或文件名。如果端口号缺省,表示使用与某种服务方式对应的标准端口号。根据查询要求不同,给出的 URL 中目录路径这一项可有可无。如果在查询中要求包括文件路径,就要在 URL 中具体指出要访问的文件名称。

下面是一些 URL 的例子:

http://www.cctv.com/	中国中央电视台网址
http://www.xjtu.edu.cn/	西安交通大学网址
ftp://ftp.xjtu.edu.cn/	西安交通大学文件服务器
gopher://gopher.xjtu.edu.cn	西安交通大学 Gopher 服务器

5. FTP 文件传输服务

文件传输协议(FTP)用来在计算机之间传输文件。Internet 是一座装满了各种资源的宝库,其中有各种软件、图片、声音、图像和动画文件,还有书籍和参考资料等。如果希望将它们下载到自己的计算机上,一个重要的方法是通过 FTP(文件传输协议)来实现,因此它是 Internet 中广为使用的一种服务。

通常,一个用户需要在 FTP 服务器中进行注册,即建立用户账号。在拥有合法的登录用户名和密码后,才有可能进行有效的 FTP 连接和登录。对于 Internet 中成千上万个 FTP 服务器来说,这就给提供 FTP 服务的管理员带来很大的麻烦,即需要为每一个使用 FTP 的用户提供一个账号,这样做显然是不现实的。实际上,Internet 的 FTP 服务是一种匿名(anonymous)FTP 服务,它设置了一个特殊的用户名,供公众使用。任何用户都可以使用这个用户名与提供这种匿名 FTP 服务的主机建立连接,并共享这个主机对公众开放的资源。

匿名 FTP 的用户名是 anonymous,而密码通常是“guest”或者是使用者的 E-mail 地址。当用户登录到匿名 FTP 服务器后,其工作方式与常规 FTP 相同。通常,出于安全的目的,大多数匿名 FTP 服务器只允许下载(Download)文件,而不允许上载(Upload)文件。也就是说,用户只能从匿名 FTP 服务器拷贝所需的文件,而不能将文件复制到匿名 FTP 服务器上。此外,匿名 FTP 服务器中的文件还加入一些保护性措施,确保这些文件不会被修改和删除,同时也可以防止计算机病毒的侵入。

FTP 是基于客户机/服务器模式的服务系统,它由客户软件、服务器软件和 FTP 通信协议 3 部分组成。FTP 客户软件运行在用户计算机上,在用户装入 FTP 客户软件后,便可以采用 FTP 通信协议,通过 FTP 内部命令与远程 FTP 服务器建立连接或传送文件。FTP 服务器软件运行在远程主机上,并设置一个名叫 anonymous 的公共用户账号,向公众开放。

FTP 在客户端与服务器内部建立两条 TCP 连接：一条是控制连接，主要用于传输命令和参数(端口号为 21)；另一条是数据连接，主要用于传送文件(端口号为 20)。FTP 服务器不断地在端口 21 上侦听用户的连接请求，当用户使用 anonymous 用户名和 guest 或者 E-mail 地址作为密码进行登录时，便是发出连接请求，这样控制连接便建立起来。此时，用户名和密码将通过控制连接发送给服务器。服务器收到这个请求后，便进行用户识别，然后向客户回送确认或拒绝的应答信息。用户看到登录成功的信息后，便可以发出文件传输的命令。服务器从控制连上收到文件名和传输命令(如 get)后，便在端口 20 发起数据连接，并在这个连接上将文件名所指定的文件传输给客户。只要用户不使用 close 或者其他命令关闭连接，便可以继续传输文件。

6. Gopher

Gopher 系统也采用客户机/服务器模式，将 Internet 上的信息组织成某种索引，很方便地将用户从一处带向另一处。Gopher 内部集成了 Telnet 和 FTP 等工具，可以直接取出文件，而无需知道文件的所在处和文件获取工具等细节。目前它的应用正在变少。

Gopher 采用一种类似于文件系统的层次结构来组织所检索的条目，并提交给用户。其工作过程为：Gopher 客户首先通过一个实体描述行来定义要检索的条目。当用户选择一个条目后，Gopher 客户软件将通过 TCP/IP 协议向指定的服务器发出建立连接的请求。如果该服务器正在已定义的 TCP 端口(端口号为 70)上侦听连接请求，则这个连接便会建立起来。然后在一个已定义的 TCP 端口上发送一个选择器字符串，指示要检索条目的路径。最后服务器将检索到的文档以文本块形式返回给客户，然后关闭连接。Gopher 协议的这种简单特性，对于在小型、低速的桌面计算机上实行客户机/服务器系统来说是快捷而有效的。

5.6 网络安全

随着网络的发展，网络的安全性显得非常重要。这是由于怀有恶意的攻击者会窃取或修改网络上传输的信息，冒充合法用户通过网络非法进入远程主机，获取存储在主机中的机密信息，或者占用网络资源，阻止其他用户使用等。这些问题的产生对网络运营部门和用户的信息安全构成了威胁，影响了计算机网络应用的进一步推广。因此，如何对计算机网络上的各种非法行为进行主动控制和有效防御，是计算机网络安全亟待解决的问题。

5.6.1 网络安全概述

计算机网络安全是指保护计算机、网络系统的硬件、软件以及系统中的数据，不因偶然的或恶意的原因而遭到破坏、更改和泄露，确保系统能连续和可靠地运行，使网络服务

不中断。从本质上来讲,网络安全就是网络上的信息安全。它涉及的领域相当广泛,这是因为目前的各种通信网络中存在着各种各样的安全漏洞和威胁。从广义来说,凡是涉及网络上信息的保密性、完整性、可用性、真实性和可控性的相关技术和理论,都是网络安全所要研究的领域。

网络的安全之所以会受到威胁,主要原因有以下几个方面:

(1) 计算机存储和处理的是有关国家安全的政治、经济、军事和国防的情况以及一些部门、机构和组织的秘密信息或个人的敏感信息和隐私,因此便成为某类人攻击的目标。

(2) 随着 Internet 的发展,存取控制和逻辑连接的数目不断地增加,软件规模不断地膨胀,很容易使系统存在缺陷。

(3) 在网络中,信息是从一台计算机的存储系统流向另一台计算机的存储系统。在大多数的情况下,信息离开信息源后必须经过多个中间节点才能到达目的地。在整个传输过程中,信息的发送者和接收者只能对发送和接收过程加以控制,而对中间传输过程则无权进行有效地控制。如果信息传输路由中存在不可信或具有攻击者的中间节点,信息的安全性就会受到严重的威胁。信息可能会被修改、破坏或泄露,因此,计算机网络的运行机制存在着严重的安全隐患。

(4) 计算机网络的运行机制是协议控制机制,不同节点之间的信息交换是按照事先定义好的固定机制,通过交换协议数据单元完成的。对于每个节点来说,通信即意味着对一系列从网络上到达的协议数据单元进行响应。根据以上的分析,这些从网上到达的协议数据单元的真实性是无法保证的。同时,由于协议本身固有的安全漏洞或协议实现中产生的安全漏洞也会造成许多安全问题。

在现有的网络环境中,由于存在不同的操作系统或不同厂家的硬件平台,故增加了网络安全的复杂性,其中有技术上和管理上的诸多原因。一个好的安全的网络应该是由主机系统、应用和服务、路由、网络、网络管理及管理制度等诸多因素决定的。系统、各种服务以及应用软件的漏洞会随着时间的加长越来越多地被发现,然后被修补。随着系统的升级又会有许多新的漏洞出现,再进行修补。因此,这是一个长期的往复循环的过程。

1) 网络安全涉及的主要内容

(1) 运行系统的安全。即保证信息处理和传输系统的安全。包括计算机系统机房环境和系统设备的保护,计算机结构设计上的安全性考虑,硬件系统的可靠与安全运行,计算机操作系统和应用软件的安全,数据库系统的安全,电磁信息泄露的防护等。

(2) 信息系统的安全。包括用户口令鉴别,用户存取权限限制和方式控制,安全审计和安全问题跟踪,计算机病毒防治,数据加密等。

(3) 信息传播的安全。信息传播后果的安全,包括信息选择和不良信息的过滤等。它侧重于防止和控制非法、有害信息传播的后果。

(4) 信息内容的安全。即狭义的信息安全。它侧重于信息的保密性、真实性和完整性。防止攻击者利用系统的安全漏洞进行窃听、冒充和诈骗等有益于合法用户的行为。本质上是保护用户的利益和隐私。

2) 信息系统对安全的基本需求

(1) 保密性。保护资源免遭非授权用户“读出”。包括传输信息的加密,存储信息的加密,防电磁泄露等。

(2) 完整性。保护资源免遭非授权用户“写入”。包括数据完整性、软件完整性、操作系统完整性、内存及磁盘完整性以及信息交换的真实性和有效性。

(3) 可用性。保护资源免遭破坏或干扰。防止病毒入侵和系统瘫痪,防止信道拥塞及拒绝服务,防止系统资源被非法抢占等。

(4) 可控性。对非法入侵提供检测与跟踪,并能干预其入侵行为的能力。

(5) 可核查性。可追查安全事故的责任人。对违反安全策略的事件提供审计手段,能记录和追踪他们的活动。

网络的安全威胁主要来自下列 5 类:

(1) 物理威胁。物理威胁是指对计算机硬件和存储介质的偷窃、废物搜寻及间谍活动。这里的废物搜寻者就像捡破烂者,不过他在废纸篓中搜寻的是机密信息。

(2) 网络攻击。计算机网络的使用对数据造成了新的安全威胁。攻击者可通过网络上的电子窃听、拨号入网和冒名顶替等方式进行入侵攻击、偷窃与篡改。

(3) 身份鉴别。由于身份鉴别通常是用设置口令的手段实现的,入侵者可通过口令圈套和密码破译等方法扰乱身份鉴别。

(4) 编程威胁。所谓编程威胁是指通过病毒进行攻击的一种方法。由于病毒是一种能进行自我复制的代码,在网间不断传播,因而更具有危害性。

(5) 系统漏洞。系统漏洞也称为系统陷阱或代码漏洞,这通常源于操作系统设计者有意设置的,目的是为了使用户在失去对系统的访问权时,仍有机会进入系统。入侵者可使用扫描器发现系统陷阱,从而进行攻击。

5.6.2 网络的信息安全

网络安全涉及到许多内容,如系统软硬件的安全,网内服务器与 PC 机使用开机热启动密码,带复杂口令的屏幕保护且启动屏保时间不要太长,尽量不要使用文件共享功能(如需要,应设置访问控制权限和密码),对重要文件设标识与验证读写口令等。严格控制管理员级账户的使用范围,经常更新系统各种密码,规定可以访问的用户数等。网络的信息安全归纳起来主要就是信息的存储安全和传输安全。

1. 信息的存储安全

信息的存储安全包括信息使用的安全(如用户的标识与验证、用户存取权限限制,安全问题跟踪等),计算机病毒防治,系统安全监控,数据的加密,防止非法的攻击等内容。

1) 用户的标识与验证

用户的标识与验证主要是限制访问系统的人员。它是访问控制的基础,是对用户身份的合法性验证。方法有两种:一是基于人的物理特征的识别,包括签名识别法、指纹识别法和语音识别法等;二是基于用户所拥有的特殊安全物品的识别,包括智能 IC 卡识别法和磁条卡识别法等。

2) 用户存取权限限制

用户存取权限限制主要是限制进入系统的用户所能执行的操作。存取控制是对各种存取活动通过授权进行控制,以保证计算机系统安全的保密机制,是对处理状态下的信息进行保护。一般有两种方法,一是隔离控制法,二是限制权限法。

隔离控制法是在电子数据处理成分的周围建立屏障,以便在该环境中实施存取规则。隔离控制技术的主要实现方式包括物理隔离方式、时间隔离方式、逻辑隔离方式以及密码技术隔离方式等。其中物理隔离方式各过程使用不同的物理目标,是一种有效的方式。传统的多网环境一般通过两台计算机实现物理隔离。现在我国已经生产出了拥有自主知识产权的涉密计算机,它采用双硬盘物理隔离技术,通过一台计算机,即可在物理隔离的状态下切换信息网和公共信息网,实现一机双网或一机多网的功能。另外一种方式就是加装隔离卡,一块隔离卡带一块硬盘和一块网卡,连同本机自带的硬盘网卡,用于不同的网络环境。当然,物理隔离方式对于系统的要求比较高,必须采用两整套互不相关的设备,其人力、物力和财力的投入都是比较大的。但也是很有效的方式,因为两者就如两条平行线,永不交叉,自然也就安全了。

限制权限法是限制进入系统的用户能够执行的操作。具体来说,就是对用户进行分类管理,安全密级授权不同的用户分在不同的类别。对目录、文件的访问控制进行严格的权限控制,防止越权操作。放置在临时目录或通信缓冲区的文件要加密,用完后尽快移走或删除。

3) 系统安全监控

系统必须建立一套安全监控系统,全面监控系统的活动,并随时检查系统的使用情况。一旦有非法入侵者进入系统,能及时发现并采取相应措施,确定和填补安全及保密的漏洞。应当建立完善的审计系统和日志管理系统,利用日志和审计功能对系统进行安全监控。管理员还应当经常做到:

- (1) 监控当前正在进行的进程,正在登录的用户情况;
- (2) 检查文件的所有者、授权、修改日期和文件的特定访问控制属性;

(3) 检查系统命令的安全配置文件、口令文件、核心启动运行文件以及任何可执行文件的修改情况;

(4) 检查用户登录的历史记录和超级用户登录的历史记录。如发现异常,及时处理。

4) 计算机病毒防治

计算机网络服务器必须加装网络病毒自动检测系统,以保护网络系统的安全,防范计算机病毒的侵袭,并且必须定期更新网络病毒检测系统。

由于计算机病毒具有隐蔽性、传染性、潜伏性、触发性和破坏性等特点,所以需要建立计算机病毒防治管理制度:

(1) 经常从软件供应商网站下载、安装安全补丁程序和升级杀毒软件。随着计算机病毒编制技术和黑客技术的逐步融合,下载、安装补丁程序和升级杀毒软件并举将成为防治病毒的有效手段。

(2) 定期检查敏感文件。对系统的一些敏感文件定期进行检查,保证及时发现已感染的病毒和黑客程序。

(3) 使用高强度的口令。尽量选择难于猜测的口令,对不同的账号选用不同的口令

(4) 经常备份重要数据。要做到每天坚持备份。较大的单位要做到每周作完全备份,每天进行增量备份,并且每个月要对备份进行校验。

(5) 选择、安装经过公安部认证的防病毒软件,定期对整个硬盘进行病毒检测和清除工作。

(6) 可以在计算机和互联网之间使用防火墙,提高系统的安全性。

(7) 当计算机不使用时,不要接入互联网,一定要断掉连接。

(8) 重要的计算机系统和网络一定要严格与互联网物理隔离。这种隔离包括离线隔离,即在互联网中使用过的系统不能再用于内网。

(9) 不要打开陌生人发来的电子邮件,无论它们有多么诱人的标题或者附件。同时也要小心处理来自熟人的邮件附件。

(10) 正确配置系统和使用病毒防治产品。正确配置系统,充分利用系统提供的安全机制,提高系统防范病毒的能力,减少病毒侵害事件。了解所选用的防病毒产品的技术特点,正确配置,以保护自身系统的安全。

5) 数据的加密

数据的加密主要是防止非法窃取或调用。包括文件信息的加密,数据库数据的安全与加密,以及磁介质的加密等。

(1) 文件信息的加密:一是文件加密,即对文件的代码或数据本身的数据源加密。二是文件名加密,就是利用文件名屏幕显示形式的变换,使得实际注册的文件名与显示的文件名不相符或者根本不显示,造成无法访问文件。

(2) 数据库数据的安全与加密: 通过对数据库系统的管理和对数据库数据的加密来实现。包括用户身份的识别和确认, 访问操作的鉴别和控制, 审计和跟踪, 数据库外加密, 数据库内加密等。

(3) 磁介质的加密: 加密的目的在于防拷贝。磁介质加密的主要方法包括固化部分程序, 激光穿孔加密, 掩膜加密和芯片加密, 修改磁介质参数表等。

6) 计算机网络安全

计算机网络安全主要是指计算机网络为抵御外界侵袭等应采取的安全措施。它是网络信息安全的最外层防线。目前主要通过采用安全防火墙系统、安全代理服务器以及安全加密网关等来实现。主要包括网络边界的安全和网络内部的安全控制与防范。

(1) 网络边界的安全。网络边界主要是指本单位(或部门)的网络与外界网络或 Internet 网互联的出口边界。其安全主要是针对经边界进出访问和传输数据包时应采取的控制和防范措施。政府各部门的计算机网络应当采用统一的国际互联网出口, 以便加强管理。计算机网络与 Internet 网或外界其他网络接入口处必须设置安全防火墙系统, 该防火墙要具有加密功能或安全加密网关。要定期扫描网络的安全漏洞, 及时消除网络安全的隐患。Internet 网或外界其他网络上的授权用户通过安全防火墙或安全加密网关远程进入政府网络时, 必须配备电子印章认证系统, 只有认证系统通过的授权用户才可进入。

计算机网络系统一般不要设置拨号访问服务器和提供远程 modem 接入, 如确需设置, 必须采用如下措施: 设置访问控制服务器, 对拨号上网的用户身份和电话号码等进行验证; 要求拨号用户采用比较安全的口令, 并确保不把用户名和口令外传给其他任何人; 在拨号访问的服务器和网络之间设置安全防火墙, 对远程访问进行控制和监测; 对拨号上网的电话号码严格保密。

(2) 网络内部的安全控制和防范。网络内部安全是指应采取防范措施以控制外界远程用户(或网络)对网络内部数据的存取。常用的技术手段包括网络安全监测报警系统、数据加/解密卡以及电子印章系统等。具体应采取以下措施: 在网络中应对信息进行相应级别的数据源加密; 对信息所需采用的各种加解密设备采用统一的加密算法和密钥管理, 同时必须定期更换密钥; 采用权限分级, 以适合不同级别的用户存取; 加装网络安全监测报警系统, 定期扫描网络的安全漏洞, 一旦有非法用户进入网络读取信息或篡改网络系统配置, 则自动报警; 必须对计算机网络用户读取信息进行身份认证, 并由此获得相应身份的读取权限, 以适应不同级别的用户读取不同级别的信息; 未经或未通过系统认证的用户, 将无权读取信息, 网络边界安全设备将禁止信息传出网络; 信息从网络边界传输出去以前, 必须经过相应的数据源加密后才能传输, 否则将无法通过网络边界的安全设备。

2. 信息的传输安全

信息的传输加密是面向线路的加密措施,有链路加密、节点加密和端到端加密3种。

(1) 链路加密:链路加密只对两个节点之间(不含信息源和目的地两个端点本身)通信线路上所传输的信息进行加密保护。它是一种链式连接的加密方式,属于公共加密。其缺点是:每个节点都要配置加密单元(信道加密机),相邻节点必须使用相同的密钥,节点的数据是明文。

(2) 节点加密:节点加密的加、解密都在节点中进行,即每个节点里都装有加、解密的保护装置,用于完成一个密钥向另一个密钥的转换。这样,节点中的数据不会出现明文。但由于每个节点都要加装安全单元或保护装置,因此需要公共网络提供配合。

(3) 端一端加密:端一端加密为系统网络提供从信息源到目的地数据传送的加密保护。可以是从主机到主机,终端到终端,终端到主机或处理进程,或从数据的处理进程到处理进程而不管数据在传送中经过了多少中间节点,数据不被解密。用户或主机都可独自采用这种加密技术而不会影响别的用户或主机,这比较适于在分组交换网中采用。

加密措施是保护信息的最后防线,被公认为是保护信息传输惟一实用的方法。在物理安全不足的地方,也是保护存储信息十分有效而经济的方法。对信息进行加密保护是在密钥的控制下,通过密码算法将敏感的机密明文数据变换成不可懂的密文数据,称为加密(Encryption)。对于一些重要的口令、数据和电子邮件用加密软件进行加密后,再发送或保存。信息即使被截获,攻击者也要耗费时间、精力去解密,从而为采取补救措施赢得了时间。

信息的传输加密技术是指发送方用加密密钥,通过加密设备或算法,将信息加密后发送出去。接收方在收到密文后,用解密密钥将密文解密,恢复为明文。如果传输中有人窃取,他只能得到无法理解的密文,从而对信息起到保密作用。基本的加密算法有两种:对称密钥加密和非对称密钥加密,用于保证电子商务中数据的保密性、完整性、真实性和非抵赖服务。

对称密钥加密也叫私有密钥加密(Private Key Encryption),即发送和接收数据的双方必须使用相同的、对称的密钥对明文进行加密和解密运算。最著名的对称密钥加密标准是数据加密标准(Data Encryption Standard,简称 DES)。DES是一种使用56个数据位的密钥来操作64位数据块的块加密算法,由IBM公司推出,可同时对大量数据进行快速加密。DES算法曾经过广泛的分析和测试,被认为是一种非常安全的系统。采用这种方法的主要问题是密钥的生成、注入、存储、管理和分发等很复杂,特别是随着用户的增加,密钥的需求量成倍增加。在网络通信中,大量密钥的分配是一个难以解决的问题。对称密钥加密的特点是简单,但用户必须让接收人知道自己使用的密钥,双方必须共同保守

密钥,任何一方的失误均会导致密钥的泄露。目前已有一些比 DES 算法更安全的对称密钥加密算法,如: IDEA 算法、RC2、RC4 算法和 Skipjack 算法等。

非对称密钥加密也叫公开密钥加密(Public Key Encryption),由美国斯坦福大学赫尔曼教授于 1977 年提出。它主要指每个人都有—对惟—对应的密钥:公开密钥和私有密钥。公钥对外公开,私钥由个人秘密保存。用其中一把密钥来加密,就只能用另一把密钥来解密。加密密钥对外是公开的,任何用户都可将传送给此用户的信息用公开密钥加密发送,而用户保存的私人密钥是保密的和惟—的,也只有它才能将密文复原、解密。虽然理论上解密密钥可由加密密钥推算出来,但这种算法设计在实际上是不可能的,或者虽然能够推算出,但要花费很长的时间而成为不可行的。所以将加密密钥公开也不会危害密钥的安全。公开密钥加密技术解决了密钥的发布和管理问题。使用公开密钥技术,进行数据通信的双方可以安全地确认对方身份和公开密钥,提供通信的可鉴别性。非对称加密算法主要有 RSA、DSA、Diffie-Hellman、PKCS 和 PGP 等。

5.6.3 防火墙技术

防火墙(Firewall)是建立在内外网络边界上的过滤封锁机制。它认为内部网络是安全和可信赖的,而外部网络是不安全和不可信赖的。防火墙的作用是防止未经授权地访问被保护的内部网络,通过边界控制强化内部网络的安全策略。它的实现有多种形式,但原理很简单。可以把它想象为一对开关,其中一个用来阻止传输,另一个用来允许传输。防火墙作为网络安全体系的基础和核心控制设备,它贯穿于受控网络通信主干线,对通过受控干线的任何通信行为进行安全处理,如控制、审计、报警和反应等,同时也承担着繁重的通信任务。由于其自身处于网络系统中的敏感位置,它还要面对各种安全威胁。因此,选用一个安全、稳定和可靠的防火墙产品,其重要性不言而喻。

在网络层,防火墙用来处理信息在内外网络边界的流动,它可以确定允许哪些地址可以传输信息或者禁止哪些目的地址的主机。在传输层,这个连接可以采用端到端的加密,也就是进程到进程的加密。在应用层,它可以进行用户级的身份认证、日志记录和账号管理等。因此简单地说法防火墙技术就是一套集身份认证、加密、数字签名和内容检查为一体的安全防范措施。所有来自 Internet 的传输信息和内部网络发出的传输信息都要穿过防火墙,由防火墙进行分析,以确保它们符合站点设定的安全策略。以提供一种内部节点或网络与 Internet 的安全屏障。

1. 防火墙的分类

防火墙技术经历了包过滤、应用代理网关和状态检测 3 个发展阶段。包过滤型的防火墙通常直接转发报文,它对用户完全透明,速度较快。应用代理网关防火墙是通过服务器建立连接的,可以有更强的身份验证和注册功能。状态检测防火墙是在其核心部分建

立状态连接表,并将进出网络的数据归为一个个会话,利用状态表跟踪每一个会话状态。状态监测不仅根据规则表对每一个包进行检查,更要考虑数据包是否符合会话所处的状态,因此提供了完整的对传输层的控制能力。

1) 包过滤型防火墙

包过滤防火墙一般有一个包检查块(通常称为包过滤器),数据包过滤可以根据数据包头中的各项信息来控制站点与站点、站点与网络以及网络与网络之间的相互访问,但无法控制传输数据的内容,因为内容是应用层数据,而包过滤器处在网络层和数据链路层(即 TCP 和 IP 层)之间。通过检查模块,防火墙能够拦截和检查所有出站和进站的数据,它首先打开包,取出包头,根据包头的信息确定该包是否符合包过滤规则,并进行记录。对于不符合规则的包,应进行报警并丢弃该包。

包过滤防火墙工作在网络层,对数据包的源及目的地 IP 地址具有识别和控制作用,对于传输层,也只能识别数据包是 TCP 还是 UDP 及所用的端口信息。由于只对数据包的 IP 地址、TCP/UDP 协议和端口进行分析,如果某条规则阻止包的传输或接收,则不允许此包通过,否则该包可以继续处理。包过滤防火墙的处理速度较快,并且易于配置。

包过滤防火墙的优点:防火墙对每条传入和传出网络的包实行低水平的控制;每个 IP 包的字段都被检查,例如源地址、目的地址、协议和端口等;防火墙可以识别和丢弃带欺骗性源 IP 地址的包;包过滤防火墙是两个网络之间访问的惟一通道;包过滤通常被包含在路由器数据包中,所以不必额外的系统来处理这个特征。

包过滤防火墙的缺点:不能完全防范黑客攻击,因为网管不可能区分出可信网络与不可信网络的界限;不支持应用层协议,因为它不认识数据包中的应用层协议,访问控制粒度太粗糙;不能处理新的安全威胁。

2) 应用代理网关防火墙

应用代理网关防火墙彻底隔断内网与外网的直接通信,内网用户对外网的访问变成防火墙对外网的访问,然后再由防火墙将外网的响应转发给内网用户。所有通信都必须经应用层代理软件转发,访问者任何时候都不能与服务器建立直接的 TCP 连接,应用层的协议会话过程必须符合代理的安全策略要求。

应用代理网关的优点是可以检查应用层、传输层和网络层的协议特征,对数据包的检测能力比较强。应用代理网关的缺点是:

(1) 难于配置。由于每个应用都要求单独的代理进程,这就要求网管能理解每项应用协议的弱点,并能合理地配置安全策略。由于配置繁琐,难于理解,容易出现配置失误,最终影响内网的安全防范能力。

(2) 处理速度非常慢。断掉所有的连接,由防火墙重新建立连接,理论上可以使应用

代理防火墙具有极高的安全性。但在实际应用中并不可行,因为对于内网的每个 Web 访问请求,应用代理都需要开一个单独的代理进程,它要保护内网的 Web 服务器、数据库服务器、文件服务器、邮件服务器及业务程序等,就需要建立一个服务代理,以处理客户端的访问请求。这样,应用代理的处理延迟会很大,内网用户的正常 Web 访问不能及时得到响应。

总之,应用代理防火墙不能支持大规模的并发连接,速度要求高的行业不能使用这类防火墙。另外,防火墙核心要求预先内置一些已知应用程序的代理,使得一些新出现的应用在代理防火墙内,不能很好地被支持。

3) 状态检测技术防火墙

状态检测技术防火墙结合了代理防火墙的安全性和包过滤防火墙的高速度等优点,在不损失安全性的基础上将代理防火墙的性能提高。

Internet 上使用的是 TCP/IP 协议, TCP 协议的每个可靠连接均需要经过“客户端同步请求”、“服务器应答”和“客户端再应答”3 次握手。例如最常用到的 Web 浏览、文件下载和收发邮件等都要经过这 3 次握手。这反映出数据包并不是独立的,前后之间有着密切的状态联系,基于这种状态变化,引出了状态检测技术。

状态检测防火墙摒弃了包过滤防火墙仅考查数据包的 IP 地址等几个参数,而不关心数据包连接状态变化的缺点,在防火墙的核心部分建立状态连接表,并将进出网络的数据看成一个个会话,利用状态表跟踪每一个会话状态。状态监测不仅根据规则表对每一个包进行检查,更考虑了数据包是否符合会话所处的状态,因此提供了完整的对传输层的控制能力。状态检测防火墙在提高安全防范能力的同时也改进了流量处理速度,因为它采用了一系列优化技术,使防火墙性能大幅度提升。因此能够应用在各种网络环境中,尤其是一些规则复杂的大型网络。

2. 典型防火墙的体系结构

一个防火墙系统通常是由过滤路由器和代理服务器组成的。过滤路由器是一个多端口的 IP 路由器,它能够拦截和检查所有出站和进站的数据。它首先打开 IP 包,取出包头,根据包头的信息(如 IP 源地址和 IP 目标地址)确定该包是否符合包过滤规则(如对包报头进行语法分析,阻止或允许包传输或接收),并进行记录。对于符合规则的包,则应进行转发,否则应进行报警并丢弃该包。代理服务防火墙使用了与包过滤器不同的方法。代理服务器使用一个客户程序与特定的中间节点(防火墙)连接,中间节点再与期望的服务器进行实际连接。与包过滤器所不同的是,使用这种类型的防火墙,内部与外部网络之间不存在直接连接。因此,即使防火墙发生了故障,外部网络也无法获得与被保护的网络的连接。代理提供了详细的注册及审计功能,这大大提高了网络的安全性,也为改进现有软件的安全性能提供了可能。它是基于特定协议的,如 FTP 和 HTTP 等。为了通过代

理支持一个新的协议,必须改进代理服务器以适应新协议。典型防火墙的体系结构包括过滤路由器、双宿主主机、被屏蔽主机以及被屏蔽子网等。

1) 包过滤路由器

包过滤路由器又称屏蔽路由器,是最简单也是最常用的防火墙。它一般作用在网络层,对进出内部网络的所有信息进行分析,并按照一定的安全策略(信息过滤规则)对进出内部网络的信息进行限制。包过滤的核心就是安全策略,即包过滤算法的设计。包过滤型防火墙往往可以用一台过滤路由器来实现,对所接收的每个数据包作出允许或拒绝的决定。如图 5-17 所示。

采用包过滤路由器的防火墙的优点在于速度快、实现方便。缺点是安全性能差。因不同操作系统环境下 TCP 和 UDP 端口号所代表的应用服务协议类型有所不同,故兼容性也差。没有或只有较少的日志记录能力。

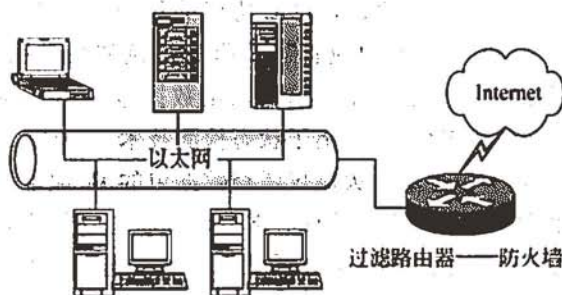


图 5-17 包过滤路由器防火墙

2) 双宿主主机

双宿主主机结构是围绕着至少具有两个网络接口的双宿主主机(又称堡垒主机)构成的,每一个接口都连接在物理和逻辑上分离的不同的网段,代理服务器软件在双宿主主机上运行,如图 5-18 所示。双宿主主机内外的网络均可与双宿主主机实施通信,但内外网络之间不能直接通信。内外网络之间的 IP 数据流被双宿主主机完全切断。结构上采用主机取代路由器实现安全控制功能。受保护网除了看到堡垒主机外,不能看到其他任何系统。另外,堡垒主机不转发 TCP/IP 通信报文,网络中的所有服务都必须由此主机的相应代理程序来支持。

双宿主主机防火墙的优势是:堡垒主机运行的系统软件可用于维护系统日志、硬件拷贝日志和远程日志等,有利于网络管理员的日后检查。其缺点是:由于它是隔开内部网和外部因特网之间的惟一屏障,若入侵者得到了双宿主主机的访问权,内部网络就会被入侵。所以为了保证内部网的安全,双宿主主机首先要禁止网络层的路由功能,同时还应具有强大的身份认证系统,尽量减少防火墙上的用户账户数。

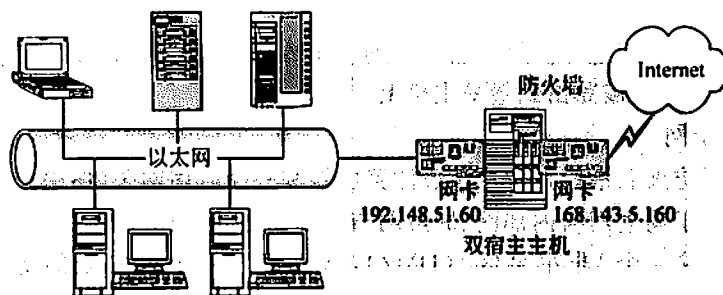


图 5-18 双宿主主机防火墙

3) 屏蔽主机网关

屏蔽主机网关防火墙由过滤路由器和应用网关组成。过滤路由器的作用是进行包过滤。应用网关的作用是代理服务,即在内部网络与外部网络之间建立两道安全屏障。屏蔽主机网关防火墙的结构如图 5-19 所示。

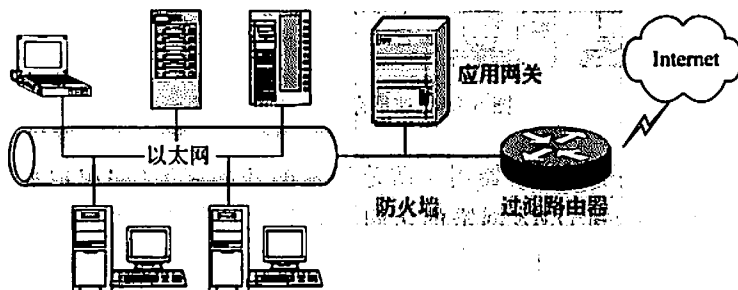


图 5-19 屏蔽主机网关防火墙

对于这种防火墙系统,堡垒主机配置在内部网络上,而包过滤路由器则放置在内部网络和 Internet 之间。在路由器上进行规则配置,使得外部系统只能访问堡垒主机,去往内部系统上其他主机的信息全部被阻塞。由于内部主机与堡垒主机处于同一个网络,内部系统是否允许直接访问 Internet,是否要求使用堡垒主机上的代理服务来访问 Internet 由机构的安全策略决定。对路由器的过滤规则进行配置,使其只接受来自堡垒主机的内部数据包,就可以强制内部用户使用代理服务。

屏蔽主机网关防火墙的优点是安全等级较高,可以提供公开信息服务的服务器,如 Web 和 FTP 等,可以放置在由包过滤路由器和堡垒主机共用的网段上。如果要求有特别高的安全特性,可以让堡垒主机运行代理服务,使内部和外部用户在与信息服务器通信之前,必须先访问堡垒主机。如果较低的安全等级已经足够,则可配置路由器,允许外部

用户直接访问公共的信息服务器。缺点是配置工作复杂。过滤路由器的配置是否正确是这种防火墙安全与否的关键。过滤路由器的路由表应当受到严格的保护,否则如果遭到破坏,则数据包就不会被路由到堡垒主机上。

4) 被屏蔽子网

被屏蔽子网防火墙系统由两个包过滤路由器和一个应用网关(堡垒主机)组成。包过滤路由器分别位于周边网与内部网、周边网与外部网之间,而应用网关居于两个包过滤路由器的中间,形成一个“非军事区”(DMZ),从而建立一个最安全的防火墙系统。如图 5-20 所示。

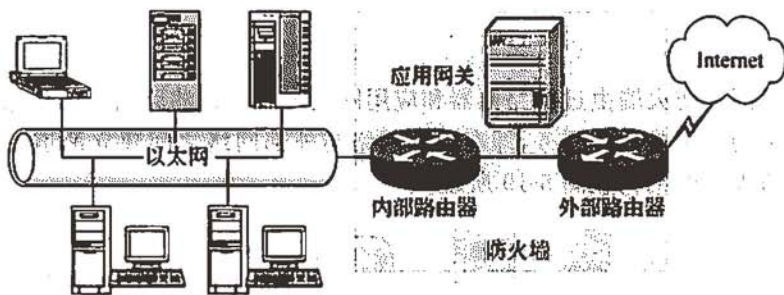


图 5-20 被屏蔽子网防火墙

对于进来的信息,外部路由器用于防范通常的外部攻击(如源地址欺骗和源路由攻击),并管理 Internet 到 DMZ 网络的访问。它只允许外部系统访问堡垒主机(还可能有信息服务器)。内部路由器提供第二层防御,它只接受源于堡垒主机的数据包,负责管理 DMZ 到内部网络的访问。对于去往 Internet 的数据包,内部路由器管理内部网络到 DMZ 网络的访问。它只允许内部系统访问堡垒主机(还可能有信息服务器)。外部路由器上的过滤规则要求使用代理服务(只接受来自堡垒主机去往 Internet 的数据包)。

被屏蔽子网防火墙系统具有下列优点:

(1) 入侵者必须突破三个不同的设备(外部路由器、堡垒主机和内部路由器)才能侵袭内部网络。

(2) 由于外部路由器只能向 Internet 通告 DMZ 网络的存在,Internet 上的系统不需要有路由器与内部网络相对应。这样网络管理员就可以保证内部网络是“不可见”的,并且只有在 DMZ 网络上选定的系统才对 Internet 开放。

(3) 由于内部路由器只向内部网络通告 DMZ 网络的存在,内部网络上的系统不能直接通往 Internet,这样就保证了内部网络上的用户必须通过驻留在堡垒主机上的代理服务才能访问 Internet。

(4) 包过滤路由器直接将数据引向 DMZ 网络上所指定的系统,消除了堡垒主机双宿的必要。

(5) 内部路由器在作为内部网络和 Internet 之间最后的防火墙系统时,能够支持比双宿堡垒主机更大的数据包吞吐量。

(6) 由于 DMZ 网络是一个与内部网络不同的网络,NAT(网络地址变换)软件可以安装在堡垒主机上,从而避免了在内部网络上重新编址或重新划分子网。

堡垒主机配置 1.1.3

net.kanto.org/privilege 为堡垒主机配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

配置路由和路由协议,需要配置路由和路由协议,配置路由和路由协议的方法如下。

第 6 章 多媒体基础知识

6.1 多媒体的基本概念

6.1.1 媒体的分类

媒体的概念范围相当广泛,按照国际电话电报咨询委员会(Consultative Committee on International Telephone and Telegraph, CCITT)的定义,媒体可以归类为下列几种。

(1) 感觉媒体(perception medium): 指直接作用于人的感觉器官,使人产生直接感觉的媒体。如引起听觉反应的声音,引起视觉反应的图像等。

(2) 表示媒体(representation medium): 指传输感觉媒体的中介媒体,即用于数据交换的编码。如图像编码(JPEG、MPEG)、文本编码(ASCII、GB2312)和声音编码等。

(3) 表现媒体(presentation medium): 指进行信息输入和输出的媒体。如键盘、鼠标、扫描仪、话筒和摄像机等为输入媒体;显示器、打印机和喇叭等为输出媒体。

(4) 存储媒体(storage medium): 指用于存储表示媒体的物理介质。如硬盘、软盘、磁盘、光盘、ROM 及 RAM 等。

(5) 传输媒体(transmission medium): 指传输表示媒体的物理介质。如电缆、光缆和电磁波等。

人们通常所说的“媒体(media)”包括两个含义。一是指信息的物理载体(即存储和传递信息的实体),如手册、磁盘、光盘、磁带以及相关的播放设备等;二是指承载信息的载体,即信息的表现形式(或者说传播形式),如文字、声音、图像、动画和视频等。上述即 CCITT 定义的存储媒体和表示媒体。表示媒体又可以分为 3 种类型:视觉类媒体(位图图像、矢量图形、图表、符号、视频和动画)、听觉类媒体(音响、语音和音乐)和触觉类媒体(点或位置跟踪,力反馈与运动反馈)。视觉和听觉类媒体是信息传播的内容,触觉类媒体是实现人机交互的手段。

多媒体就是指多种信息载体的表现形式和传递方式。通常多媒体是对多种媒体的融合,即将音频、视频、图像和计算机技术与通信技术集成到同一数字环境中,以协同表示更丰富和复杂的信息。为了使不同的媒体能够有机地连接起来,往往应按照实际需要建立一种链接机制或结构,这种链接机制或结构被称为超媒体(hyper media)。在计算机中,超媒体是一个信息存储和检索系统,它把文字、图形、图像、动画、声音和视频等媒体集成

为一个相关的基本信息系统。通常,如果信息主要以文字的形式表示,则称其为超文本。如果还包含图形、影视、动画、音乐或其他媒体,一般称其为超媒体。互联网的 WWW 应用是超媒体技术的最好例子。

6.1.2 多媒体的特征

在计算机领域中,“多媒体”常常被当作“多媒体技术”的同义词。多媒体技术就是指利用计算机技术把文本、图形、图像、声音、动画和视频等多种媒体综合起来,使多种信息建立逻辑连接,并能对它们进行获取、压缩、加工处理及存储,集成为一个具有交互性的系统。多媒体技术的内涵和范围极其广泛,所涉及的技术也极为广泛,其主要特性如下所述。

(1) 多样性:主要表现在信息媒体的多样化。多样性使得计算机处理的信息空间范围扩大,不再局限于数值、文本或特殊对待的图形和图像。可以借助于视觉、听觉和触觉等多种感觉形式实现信息的接收、产生和交流。

(2) 集成性:主要表现在多媒体信息(文字、图形、图像、语音和视频等信息)的集成和操作这些媒体信息的软件和设备的集成。多媒体信息的集成是将各种信息媒体按照一定的数据模型和组织结构集成为一个有机的整体。操作媒体信息的软件和设备集成是将与多媒体相关的各种硬件和软件集成为一个理想的环境,以便充分共享和使用多媒体信息。

(3) 交互性:这是多媒体应用有别于传统信息交流媒体的主要特点之一。传统信息交流媒体只能单向地、被动地传播信息,而多媒体技术引入交互性后则可实现人对信息的主动选择、使用、加工和控制。通过交互与反馈,可以更加有效地控制和使用信息,为人们提供发挥创造力的环境,增强了人们的参与感,同时也为多媒体技术的应用开辟了更加广阔的领域。

(4) 非线性:多媒体技术的非线性特点将改变人们传统的循序性读写模式。以往人们的读写方式大都采用章、节和页的框架,循序渐进地获取知识。而多媒体技术将借助超文本链接的方法,把内容以一种更灵活、更具变化的方式呈现给读者。

(5) 实时性:是指在人的感官系统允许的情况下进行多媒体处理和交互。当人们给出操作命令时,相应的多媒体信息都能够得到实时控制。

(6) 信息使用的方便性:用户可以按照自己的需要、兴趣、任务要求、偏爱和认知特点来使用信息,获取图、文及声等信息表现形式。

(7) 信息结构的动态性:用户可以按照自己的目的和认知特征重新组织信息,即增加、删除或修改节点,重新建立链接等。

6.2 音频

6.2.1 数字声音基础

1. 声音信号

声音是通过空气传播的一种连续的波,称为声波。声波在时间和幅度上都是连续的模拟信号,通常称为模拟声音(音频)信号。人们对声音的感觉主要有音量、音调和音色3个指标。

(1) 音量(也称响度):声音的强弱程度,取决于声音波形的幅度,即取决于振幅的大小和强弱。

(2) 音调:人对声音频率的感觉表现为音调的高低,取决于声波的基频。基频越低,给人的感觉越低沉,频率高则声音尖锐。

(3) 音色:由混入基音(基波)的泛音(谐波)所决定,每种声音都有其固定的频率和不同音强的泛音,从而使得它们具有特殊的音色效果。人们能够分辨具有相同音高的钢琴和小号声音,就是因为它们具有不同的音色。一个声波上的谐波越丰富,音色越好。

对声音信号的分析表明,声音信号由许多频率不同的信号组成,通常称为复合信号。而把单一频率的信号称为分量信号。声音信号的一个重要参数就是带宽(bandwidth),它用来描述组成声音的信号的频率范围。PC机处理的音频信号主要是人耳能听得到的音频信号(audio),它的频率范围是20Hz~20kHz。可听声包括:

(1) 话音(也称语音) 人的说话声,频率范围通常为300Hz~3400Hz。

(2) 音乐 由乐器演奏形成(规范的符号化声音),其带宽可达到20Hz~20kHz。

(3) 其他声音 如风声、雨声、鸟叫声和汽车鸣笛声等,它们起着效果声或噪声的作用,其带宽范围也是20Hz~20kHz。

声音信号的两个基本参数是幅度和频率。幅度指声波的振幅,通常用动态范围表示,一般用分贝(dB)为单位来计量。频率指声波每秒钟变化的次数,用Hz表示。人们把频率小于20Hz的声波信号称为亚音信号(也称次音信号),频率范围为20Hz~20kHz的声波信号称为音频信号,高于20kHz的信号称为超音频信号(也称超声波)。

2. 声音信号的数字化

声音信号是一种模拟信号,计算机要对它进行处理,必须将它转换成数字声音信号,即用二进制数字的编码形式来表示声音。最基本的声音信号数字化方法是取样-量化法,它分成如下3个步骤。

① 采样:采样是把时间连续的模拟信号转换成时间离散、幅度连续的信号。在某些

特定时刻获取的声音信号幅值叫做采样,由这些特定时刻的采样得到的信号称为离散时间信号。一般都是每隔相等的一小段时间采样一次,其时间间隔称为取样周期,它的倒数称为采样频率。采样定理是选择采样频率的理论依据,为了不产生失真,采样频率不应低于声音信号最高频率的两倍。因此,语音信号的采样频率一般为 8kHz,音乐信号的采样频率则应在 40kHz 以上。采样频率越高,可恢复的声音信号分量越丰富,其声音的保真度越好。

② 量化:量化处理是把在幅度上连续取值(模拟量)的每一个样本转换为离散值(数字量)表示,因此量化过程有时也称为 A/D 转换(模数转换)。量化后的样本是用二进制数来表示的,二进制数位数的多少反映了量度声音波形幅度的精度,称为量化精度,也称为量化分辨率。例如,每个声音样本若用 16 位(2B)表示,则声音样本的取值范围是 0~65 536,精度是 1/65 536。若只用 8 位(1B)表示,则样本的取值范围是 0~255,精度是 1/256。量化精度越高,声音的质量越好,需要的存储空间也越多。量化精度越低,声音的质量越差,但需要的存储空间也少。

③ 编码:经过采样和量化处理后的声音信号已经是数字形式了,但为了便于计算机存储、处理和传输,还必须按照一定的要求进行数据压缩和编码,即选择某一种或者几种方法对它进行数据压缩,以减少数据量,再按照某种规定的格式将数据组织成为文件。

经过数字化处理之后的数字声音的主要参数参见表 6-1。

表 6-1 数字化处理之后的数字声音的主要参数

参数	说 明
采样频率	表示每秒钟内采样的次数,采样的 3 个标准频率分别为 44.1kHz、22.05 kHz 和 11.05 kHz
量化位数	反映量度声音波形幅度的精度,声音信号的量化精度一般为 8 位、12 位或 16 位
声道数目	单声道一次产生一组声音波形数据,双声道则一次同时产生两组声音波形数据
数据率	表示每秒钟的数据量,以 Kb/s 为单位
压缩比	同一时间间隔内的音频数据压缩前的数据量与压缩后的数据量之比。压缩比通常大于 1

3. 个人计算机中的声音表示方法

个人计算机中的数字声音有两种不同的表示方法。一种称为波形声音(也称为自然声音),通过对实际声音的波形信号进行数字化(取样和量化)而获得,它能高保真地表示现实世界中任何客观存在的真实声音。例如,44.1kHz×16b 的 CD 质量的声音,8kHz×8b 的数字语音等。波形声音的数据量比较大。另一种是“合成声音”,它使用符号(参数)对声音进行描述,然后通过合成(synthesize)的方法生成声音。例如,MIDI 音乐(用符号描述的乐器演奏的音乐声音)、合成语音(用声母、韵母或清音、基音频率等参数描述的语音)等。符号化的声音表示方法所产生的声音虽然没有自然声那么真实和逼真,但数据量

要比波形声音小得多(2~3 个数量级),而且能产生自然界中不存在的声音,其编辑处理也比波形声音更加方便一些。

声音信息的计算机处理比数值信息和文本信息的处理要复杂一些,以波形声音的编辑处理为例,常见的基本编辑操作有声音的分段、拼接、音量调整、降低采样频率、均匀化、时间扩展以及声音的各种效果处理(如颤抖、延时、混响和音调变换)等。

个人计算机和多媒体系统对数字声音的处理是与应用密切相关的,涉及到多方面的声音信息处理技术,大体可以分为:

- 声音的获取、重建与播放;
- 数字声音的编辑处理;
- 数字声音的存储与检索;
- 数字声音的传输;
- 数字语音与文本的相互转换等。

6.2.2 波形声音

波形声音信息是一个用来表示声音振幅的数据序列,它是通过对模拟声音按一定间隔采样获得的幅度值,再经过量化和编码后得到的便于计算机存储和处理的数据格式。声音信号数字化后,其数据传输率(每秒比特数)与信号在计算机中的实时传输有直接关系,而其总数据量又与计算机的存储空间有直接关系。未经压缩的数字音频的数据传输率可按下式计算:

$$\text{数据传输率(b/s)} = \text{采样频率(Hz)} \times \text{量化位数(b)} \times \text{声道数}$$

其中,数据传输率以每秒比特(b/s)为单位,采样频率以 Hz 为单位,量化以比特(b)为单位。

波形声音经过数字化后所需占用的存储空间可用如下公式计算:

$$\text{声音信号数据量} = \text{数据传输率} \times \text{持续时间} / 8(\text{B})$$

【例 6.1】 若语音信号的带宽为 300Hz~3400Hz,量化精度为 8b,单声道输出,计算每秒钟及每小时的数据量。

解:根据题意,数字化时的取样频率为 8kHz,根据上述公式每秒钟的数据量为:

$$\text{采样频率} \times \text{量化位数} \times \text{声道数} / 8 = 8000 \times 8 \times 1 / 8 = 64 \text{Kb/s} = 8 \text{KB/s}$$

1 小时数字语音的数据量大约是 28MB。

【例 6.2】 若 CD(compact disc,光盘)盘片上所存储的立体声高保真数字音乐的带宽为 20Hz~20 000Hz,采样频率为 44.1kHz,量化精度为 16 位,双声道,计算 1 小时的数据量。

解:每秒钟的数据量为 1411.2Kb/s(176.4KB/s),1 小时的数据量大约是 635MB。

可见,数字波形声音数据量非常大,因此在编码的时候常常要采用压缩的方式来压缩数字数据,以减少存储空间和提高传输效率(降低传输带宽)。根据统计分析结果,语音信号中含有大量的冗余信息,再加上还可以利用人的听觉感知特性,因此产生了许多能满足实际应用需求的压缩算法。一个好的数据压缩算法通常应能满足下列需求:

- 压缩倍数高,压缩后的数据率低;
- 解码后的信号失真小,质量高;
- 算法简单,执行速度快,延迟时间短;
- 编码器/解码器的成本低。

数字语音的数据压缩方法很多,从原理上可分为 3 类。

(1) 波形编码:这是一种直接对取样量化后的波形进行压缩处理的方法。例如脉冲编码调制(PCM)、自适应差分脉冲编码(ADPCM)以及子带编码(SBC)等。波形编码的特点是通用性强,不仅适用于数字语音的压缩,而且对所有使用波形表示的数字声音都有效。可获得高质量的语音,但很难达到很高的压缩比。

(2) 参数编码(也称为模型编码):这是一种基于声音生成模型的压缩方法(production model-based compression),从语音波形信号中提取生成的语音参数,使用这些参数通过语音生成模型重构出语音。例如线性预测编码(LPC)和声码器(vocoder)等。它的优点是能达到很高的压缩比。缺点是信号源必须已知,而且受声音生成模型的限制,质量还不理想。

(3) 混合编码:波形编码虽然可提供高质量的语音,但数据率比较高,很难低于 16Kb/s。参数编码的数据率虽然可降低到 3Kb/s,甚至更低,但它的音质根本不能与波形编码相比。混合编码是上述两种方法的结合,它既能达到高的压缩比,又能保证一定的质量。但算法相对复杂一些,例如码激励线性预测(CELP)和混合激励线性预测(MELP)等。

基于波形的压缩编码方法虽然数据率比较高,但能保证高质量地重建语音,且算法简单、易实现,能较好地保持原有声音的特点,便于编辑处理,因而在多媒体计算机和多媒体文档中有广泛的应用。

数字语音压缩编码有多种国际标准,如 G. 711、G. 721、G. 726、G. 727、G. 722、G. 728、G. 729A、G. 723. 1 和 IS96(CDMA)等。

数字语音由于频带比较窄,又可以通过语音生成模型进行比较好的模拟,因此经过压缩编码后码率比较低,存储与传输问题不大。而对于有高保真度要求的全频带声音,由于其带宽达到 20kHz 以上,又有双声道甚至多声道的要求,数据量相当客观,对压缩编码的要求更高。全频带声音的压缩编码方法与数字语音的处理方法不同,它不但依据波形本身的相关性,而且还利用人的听觉系统特性来达到压缩声音的目的,这种压缩编码称为感

知声音编码(perceptual audio coding)。在国际标准 MPEG 中,先后为视频图像伴音的数字宽带声音制定了 MPEG—1 Audio、MPEG—2 Audio、MPEG—2AAC 和 MPEG—4 Audio 等多种数据压缩编码的标准。MPEG 处理的是 10Hz~20 000Hz 频率范围的声音信号,数据压缩的主要依据是人耳的听觉特性,特别是人耳存在着随声音频率变化的听觉域,以及人耳的听觉掩蔽特性。

6.2.3 声音合成

个人计算机和多媒体系统中的声音,除了数字波形声音之外,还有一类是使用符号表示的。由计算机合成的声音,包括语音合成和音乐合成。

1. 语音合成

语音合成目前主要指从文本到语音的合成,也称为文语转换。采用文语转换的方法输出语音,应预先建立语音参数数据库、发音规则库等。需要输出语音时,系统按需求先合成语音单元,再按语音学规则或语言学规则,连接成自然的语流。文语转换的参数数据库不随发音时间加长而增大,但规则库却随语音质量的要求而增大。语音合成有多方面的应用,例如,海量查询与声讯服务、航班动态查询、电话报税和有声 E-mail 等,这些业务都需要以准确、清晰的语音通过电话进行操作提示并提供查询结果。一般来说,对合成的语音要求能够做到:可听懂、自然、延迟时间短以及速度可控制等。

文语转换原理上一般分成两步,第一步先将文字序列转换成音韵序列,第二步再由语音合成器生成语音波形(见图 6-1)。其中第一步涉及语言学处理,例如分词、字音转换等,以及一整套有效的韵律控制规则。第二步需要使用语音合成技术,能按要求实时合成出高质量的语音流。

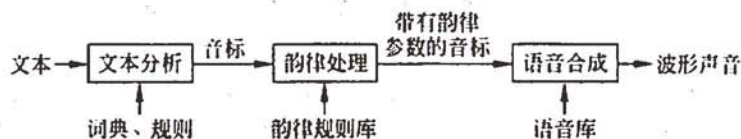


图 6-1 文语转换过程示意

从合成采用的技术来说,语音合成可分为发音参数合成、声道模型参数合成和波形编辑合成。

(1) 发音参数合成:这种方法对人的发音过程进行直接模拟,它定义了唇、舌和声带的相关参数,如开口度、舌高度、舌位置和声带张力等。由这些发音参数估计声道截面积函数,进而计算声波。但由于人的发音生理过程比较复杂,以及理论计算与物理模拟之间存在较大差异,合成语音的质量目前还不理想。

(2) 声道模型参数合成: 这种方法基于声道截面积函数或声道谐振特性合成语音, 如共振峰合成器和 LPC 合成器。这类合成器的比特率低, 音质适中。为改善音质, 发展了混合编码技术, 主要手段是改善激励, 如码本激励、多脉冲激励以及长时预测规则码激励等。这样比特率有所增大, 同时音质得到提高。

(3) 波形编辑合成: 波形编辑合成技术是直接吧语音波形数据库中的波形相互拼接在一起, 输出连续语流。这种语音合成技术用原始语音波形代替参数, 而且这些语音波形取自自然语言的词或句子, 它隐含了声调、重音和发音速度的影响, 合成的语言清晰自然。其质量普遍高于参数合成。

2. 音乐合成

音乐是用乐谱进行描述, 由乐器演奏而成的。乐谱的基本组成单元是音符(notes), 最基本的音符有 7 个, 所有不同音调的音符少于 128 个。

音符代表的是音乐, 音乐与噪声的区别主要在于它们是否有周期性。音乐的波形随着时间作周期性变化, 噪声则不然。音乐的要素有音调、音色、响度和持续时间。音调指声波的基频, 基频低, 声音低沉, 基频高, 声音高昂。不同乐器有不同的音色, 音色是由声音的频谱决定的。响度即声音的强度。一首乐曲中每一个乐音的持续时间是变化的, 从而形成旋律。音乐可以使用电子学原理合成出来(生成相应的波形), 各种乐器的音色也可以进行模拟。电子乐器由演奏控制器和音源两部分组成:

(1) 演奏控制器: 演奏控制器是一种输入和记录实时乐曲演奏信息的设备。它的作用是像传统乐器那样用于演奏, 驱动音源发声, 同时它也是计算机音乐系统的输入设备。其类型有键盘、气息(呼吸)控制器和弦乐演奏器等。

(2) 音源(也称为音乐合成器): 音源是具体产生声音波形的部分, 即电子乐器的发声部分。它通过电子线路把演奏控制器送来的声音合成起来。最常用的音源有两类:

- 数字调频合成器(FM): FM 是使高频振荡波的频率按调制信号规律变化的一种调制方式。在音乐合成器中, 数字载波波形和调制波形有很多种。采用不同的调制波频率和调制指数, 就可以方便地合成具有不同频谱分布的波形, 再现某些乐器的音色。可以使用这种方法得到具有独特效果的电子模拟声, 创造出丰富多彩的, 并且是真实乐器所不具备的音色。
- PCM 波形合成器(波表合成法): 使用 FM 合成法来产生逼真的声音不够理想, 有些音乐几乎不能产生, 因此就自然地转而采用 PCM 波形合成法, 即波表合成法。这种方法能把真实乐器发出的声音以数字形式记录下来, 将它们放在一个波形表中, 合成音乐时以查表匹配方式获取真实的乐器波形。这种以真实声音波形为基础的音源音色真实, 音质丰满, 合成的音乐基本上能达到以假乱真的效果。但是, 由于采样波形不能代表所有真实的演奏状态, 而且音乐的音色在某种程度上还取

决于演奏者的演奏技术,因而还是达不到演奏者演奏音乐的临场感觉效果。

6.2.4 MIDI

MIDI(musical instrument digital interface)是乐器数字接口的缩写,泛指数字音乐的国际标准。MIDI 标准规定了电子乐器与计算机之间连接的电缆硬件,以及电子乐器之间、乐器与计算机之间传送数据的通信协议。规定了音乐的数字表示,包含音符、定时和乐器指派等规范。由于 MIDI 标准定义了计算机音乐程序、合成器及其他电子设备交换信息和电子信号的方式,所以可以解决不同电子乐器之间不兼容的问题。符合 MIDI 规范的设备称为 MIDI 设备。通过 MIDI 接口,不同 MIDI 设备之间可进行信息交换。带有 MIDI 接口以及专用 MIDI 电缆的计算机、合成器和其他 MIDI 设备连接在一起,即可构成计算机音乐系统。数据由 MIDI 设备的键盘产生,可通过声音合成器还原为声音。通过计算机可以控制乐器的输出,并能接收、存储和处理经过编码的音乐数据。

MIDI 数据不是单个采样点的编码(波形编码),而是乐谱的数字描述,称为 MIDI 消息。乐谱由音符序列、定时、音色和音量等组成。每个消息对应一个音乐事件(如键按下、键释放等)。一组 MIDI 消息送到 MIDI 音源时,音源即合成出相应的音乐。

MIDI 文件是计算机中用于存储和交换 MIDI 消息的一种数据文件,它由一系列 MIDI 消息组成。它不仅包含与音乐演奏有关的消息,如音符、速度和表情等信息,而且还包含大量与音响有关的控制信息,如声像、效果和均衡等。MIDI 文件中包含多达 16 个通道的演奏定义,包括每个通道的演奏音符信息,如键、音长、音量和力度(击键时,键到达最低位置的速度)等。标准 MIDI 文件格式采用的文件扩展名为 mid,它是音序软件的文件交换标准,也是商业音乐作品发行的标准。

音序器(sequencer)又称声音序列发生器,有硬件产品,也有软件产品(也称音乐软件或作曲软件)。它具有丰富的编辑和存储功能。首先,音序器可将演奏者实时演奏的音符(note)、节奏信息以及各种表情控制信息,如速度、触键力度、颤音以及音色变化等以数字方式,在计算机时钟的基础上,按时间或节拍顺序记录下来,然后对记录下来的信息进行修改和编辑,经过编辑修改的演奏信息在任意时刻都可以发送给音源,音源可自动演奏播放。这里关键是要实现“分轨录音”,而这些属于不同声部的演奏信息可被音序器记录在不同的 MIDI 通道中。通过音源,音序器可将所有 MIDI 通道中的演奏信息同时自动播放演奏。这样,一个人就可完成相当于一个乐队的多声部演奏和录音任务。以音序器软件为核心的计算机个人音乐系统,彻底改变了传统的音乐制作方式和概念,原来需要由多人才能完成的工作现在只需一个人即可。记录音乐的方式也由原来的乐谱变成了 MIDI 文件,音乐作品由修改困难变为可进行任意的编辑修改。

MIDI 音乐与高保真的波形声音相比,虽然在音质方面还有一些差距,也无法合成出

所有各种不同的声音(如语音),且生成的音乐质量与使用的音源硬件有关,但它的数量级极少(比 CD-DA 少 3 个数量级,比 MP3 少 2 个数量级),又易于编辑修改,还可以与波形声音同时播放。

6.2.5 声音文件格式

数字声音在计算机中存储和处理时,其数据必须以文件的形式进行组织,所选用的文件格式必须得到操作系统和应用软件的支持。如同文本文件一样,在互联网上、各种不同计算机以及应用软件中使用的声音文件格式也互不相同。下面通过声音文件的特征后缀名来逐一介绍。

(1) Wave 文件(.WAV): Microsoft 公司的音频文件格式,它来源于对声音模拟波形的采样。用不同的采样频率对声音的模拟波形进行采样可以得到一系列离散的采样点,以不同的量化位数(8 位或 16 位)把这些采样点的值转换成二进制数,然后存入磁盘,这就产生了声音的 WAV 文件,即波形文件。利用该格式记录的声音文件能够和原声基本一致,质量非常高,但文件数据量大。

(2) Module 文件(.MOD): 该格式的文件里存放乐谱和乐曲使用的各种音色样本,具有回放效果较好,音色种类无限等优点。

(3) MPEG 音频文件(.MP3): 是现在最流行的声音文件格式,因其压缩率大,在网络可视电话通信方面应用广泛,但和 CD 唱片相比,音质仍然不能令人非常满意。

(4) RealAudio 文件(.RA): 这种格式具有强大的压缩量和较小的失真,它也是为了解决网络传输带宽资源而设计的,因此其主要目标是压缩比和容错性,其次才是音质。

(5) MIDI 文件(.MID/.RMI): 它是目前较成熟的音乐格式,实际上已经成为一种产业标准,其科学性、兼容性和复杂程度等各方面当然远远超过本文前面介绍的所有标准(除交响乐 CD、Unplug CD 外,其他 CD 往往都是利用 MIDI 制作出来的),General MIDI 就是最常见的通行标准。作为音乐工业的数据通信标准,MIDI 能指挥各音乐设备的运转,而且具有统一的标准格式。能够模仿原始乐器的各种演奏技巧甚至无法演奏的效果,而且文件的长度非常小。RMI 可以包括图片标记和文本。

(6) Voice 文件(.VOC): Creative 公司波形音频文件格式,也是声霸卡(sound blaster)使用的音频文件格式。每个 VOC 文件由文件头块(header block)和音频数据块(data block)组成。文件头包含一个标识版本号和一个指向数据块起始位置的指针。数据块分成各种类型的子块,如声音数据静音标识 ASCII 码文件重复的结果重复以及终止标志、扩展块等。

(7) Sound 文件(.SND): Sound 文件是 NeXT Computer 公司推出的数字声音文件格式,支持压缩。

(8) Audio 文件(. AU): Audio 文件是 Sun Microsystems 公司推出的一种经过压缩的数字声音文件格式,是互联网上常用的声音文件格式。

(9) AIFF 文件(. AIF): Apple 计算机的音频文件格式。Windows 的 Convert 工具可以把 AIF 格式的文件换成 Microsoft 的 WAV 格式的文件。

(10) CMF 文件(. CMF): Creative 公司的专用音乐格式,与 MIDI 差不多,音色、效果上有些特色,专用于 FM 声卡,兼容性较差。

6.3 图形和图像

6.3.1 色彩与图像基础

1. 色彩的基本概念

色彩是创建图像的基础,在计算机上使用色彩没有什么特殊之处,只不过是它有一套特定的记录和处理色彩的技术。因此,要理解图像处理软件中出现的各种有关色彩术语,首先要具备基本的色彩理论知识。

色彩是通过光被人们感知的。物体由于内部物质的不同,受光线照射后,产生光的分解现象,一部分光线被吸收,其余的被反射或投射出来,成为人们所见的物体的色彩。所以,色彩和光有密切关系,同时还与被光照射的物体有关,并与观察者有关。

彩色光作用于人眼,使之产生彩色视觉。为了能确切地表示某一彩色光的量度,可以用亮度、色调和色饱和度 3 个物理量来描述,并称之为色彩三要素。

(1) 亮度:亮度是描述光作用于人眼时引起的明暗程度感觉,是指彩色明暗深浅程度。一般说来,对于发光物体,彩色光辐射的功率越大,亮度越高;反之,亮度越低。对于不发光的物体,其亮度取决于吸收或者反射光功率的大小。

(2) 色调:色调指颜色的类别。所谓的红色、绿色和蓝色等不同颜色就是指色调。由光谱分析可知,不同波长的光呈现不同的颜色,人眼看到一种或多种波长的光时所产生的彩色感觉,反映出颜色的类别。某一物体的色调取决它本身辐射的光谱成分,或在光的照射下所反射的光谱成分对人眼刺激的视觉反应。

(3) 色饱和度:色饱和度是指某一颜色的深浅程度(或浓度)。对于同一种色调的颜色,其饱和度越高,则颜色越深,如深红、深绿和深蓝等。其饱和度越低,则颜色越淡,如淡红、淡绿和淡黄等。高饱和度的深色光掺入白色光可以被冲淡,降为低饱和度的淡色光。因此,色饱和度可看作某色调的纯色掺入白色光的比例。例如,一束高饱和度的蓝色光投射到屏幕上会被看成深蓝色光,若再将一束白色光也投射到屏幕上并与深蓝色重叠,则深蓝色变成淡蓝色。投射的白色光越强,颜色越淡,即饱和度越低。相反,由于彩色电视屏

幕上的亮度过高,使色饱和度降低,颜色被冲淡。这时可以降低亮度(白光)从而使色饱和度和度增大,颜色加深。

2. 三基色原理

从理论上讲,任何一种颜色都可以用3种基本颜色按不同比例混合得到。自然界常见的各种颜色光,都可由红(Red)、绿(Green)和蓝(Blue)3种颜色光按不同比例相配而成。同样,绝大多数颜色光也可以分解成红、绿和蓝3种颜色光,这就是色度学中最基本的三基色原理。当然,三基色的选择不是惟一的,可以选择其他3种颜色为三基色。但是,3种颜色必须是相互独立的,即任何一种颜色都不能由其他两种颜色合成。由于人眼对红、绿和蓝3种颜色光最敏感,因此由这3种颜色相配所得的彩色范围也最广,所以一般都选用这3种颜色作为基色。把3种基色光按不同比例相加称之为相加混色,由红、绿和蓝三基色进行相加混色的情况如下:

红色+绿色=黄色

红色+蓝色=品红

绿色+蓝色=青色

红色+绿色+蓝色=白色

红色+青色=绿色+品红=蓝色+黄色=白色

如果两种色光混合而成白光,则这两种色光互为补色。

3. 彩色空间

彩色空间指彩色图像所使用的颜色描述方法,也称为彩色模型。在PC机和多媒体系统中,表示图形和图像的颜色常常涉及不同的彩色空间,如RGB彩色空间、CMY彩色空间以及YUV彩色空间等。不同的彩色空间对应不同的应用场合,各有其特点。因此,数字图像的生成、存储、处理及显示时对应不同的彩色空间。从理论上讲,任何一种颜色都可以在上述彩色空间中精确地进行描述。

(1) RGB彩色空间: 计算机中的彩色图像一般都用R、G、B分量表示,彩色显示器通过发射出3种不同强度的电子束,使屏幕内侧覆盖的红、绿和蓝荧光材料发光而产生色彩。这种彩色的表示方法称为RGB彩色空间表示法。因为彩色显示器的输入需要R、G、B彩色分量,通过3个分量的不同比例,在显示屏幕可合成任意需要的颜色。无论多媒体系统中间过程采用什么形式的彩色空间表示,最后的输出一定要转换成RGB彩色空间表示。

(2) CMY彩色空间: 在RGB彩色空间中,不同颜色的光是通过相加混合实现的。而彩色打印的纸张是不能发射光线的,因而彩色打印机就不能采用RGB颜色来打印,它只能使用能够吸收特定光波而反射其他光波的油墨或颜料来实现。用油墨或颜料进行混合得到的彩色称为相减混色。之所以称为相减混色,是因为它减少(吸收)了人眼识别颜

色所需要的反射光。根据三基色原理,油墨或颜料的三基色是青(cyan)、品红(magenta)和黄(yellow)。可以用这3种颜色的油墨或颜料按不同比例混合成任何一种颜色,这种彩色表示方法称为CMY彩色空间。

(3) YUV彩色空间:在现代彩色电视系统中,通常采用三管彩色摄像机或彩色CCD摄像机,把摄得的彩色图像信号,经过分色、放大和校正得到R、G、B三基色,再经过矩阵变换得到亮度信号Y、色差信号U(R-Y)和V(B-Y),最后发送端将这三个信号分别进行编码,用同一信道发送出去。这就是通常用的YUV彩色空间。电视图像一般都采用Y、U、V分量表示,即一个亮度分量(Y)和两个色差(U和V)分量表示。由于亮度和色度是分离的,因而解决了彩色和黑白显示系统的兼容问题。如果只有Y分量而没有U、V分量,那么所表示的图像就是黑白灰度图像。

6.3.2 计算机中的图形数据表示

计算机中的图形数据有两种常用的表示形式,一种称为几何图形或矢量图形,简称图形,另一种称为点阵图像或位图图像。

1. 矢量图形

矢量图形是用一系列计算机指令来描述和记录的一幅图的内容,即通过指令描述构成一幅图的所有直线、曲线、圆、圆弧和矩形等图元的位置、维数和形状,也可以用更为复杂的形式表示图像中的曲面、光照和材质等效果。矢量图法实质上是用数学的方式(算法和特征)来描述一幅图形图像。在处理图形图像时,根据图元对应的数学表达式进行编辑和处理。在屏幕上显示一幅图形图像时,首先要解释这些指令,然后将描述图形图像的指令转换成屏幕上显示的形状和颜色。编辑矢量图的软件通常称为绘图软件,如适于绘制机械图、电路图的AutoCAD软件等。这种软件可以产生和操作矢量图的各个成分,并对矢量图形进行移动、缩放、叠加、旋转和扭曲等变换。编辑图形时将指令转变成屏幕上所显示的形状和颜色,显示时也往往能看到绘图的过程。由于所有的矢量图形部分都可以用数学的方法加以描述,从而使得计算机可以对其任意进行放大、缩小、旋转、变形、扭曲、移动和叠加等变换,而不会破坏图像的画面。但是,用矢量图形格式表示复杂图像(如人物、风景照片),并且要求很高时,将需要花费大量的时间进行变换、着色以及处理光照效果等。因此,矢量图形主要用于表示线框型的图画、工程制图和美术字等。多数CAD和3D造型软件使用矢量图形作为基本的图形存储格式。

2. 位图图像

位图图像是指用像素点描述的图。图像一般是用摄像机或扫描仪等输入设备捕捉实际场景画面,离散化为空间、亮度和颜色(灰度)的序列值,即把一幅彩色图或灰度图分成许许多多的像素(点),每个像素用若干二进制位来指定该像素的颜色、亮度和属性。位图

图像在计算机内存中由一组二进制位(bit)组成,这些位定义图像中每个像素点的颜色和亮度。屏幕上一个点也称为一个像素,显示一幅图像时,屏幕上的一个像素也就对应于图像中的某一点。根据组成图像的像素密度和表示颜色、亮度级别的数目,又可将图像分为二值图(黑白图)和彩色图两大类,彩色图还可以分为真彩色图和伪彩色图等。位图适合于表现比较细腻、层次较多、色彩较丰富以及包含大量细节的图像,并可直接、快速地在屏幕上显示出来。但占用存储空间较大,一般需要进行数据压缩。

6.3.3 图像的获取

将现实世界的景物或物理介质上的图文输入计算机的过程称为图像的获取(capturing)。多媒体应用中的基本图像可通过不同的方式获得。一般来说,可以直接利用数字图像库的图像,可以利用绘图软件创建图像,也可以利用数字转换设备采集图像。

1. 利用数字图像库

在 CD-ROM 光盘上和互联网上的图像库中,图像的内容较丰富,图像尺寸和图像深度可选的范围也较广。图像的质量完全可以满足一般用户的要求,但图像的内容也许不符合用户的创意设计。用户可根据需要选择已有的图像,或再做进一步的编辑和处理。

2. 利用绘图软件创建图像

目前 Windows 环境的大部分图像编辑软件都具有一定的绘图功能。这些软件大多具有较强的功能和很好的图形用户接口。还可以利用鼠标、画笔及画板来绘制各种图形,并进行彩色、纹理和图案等的填充和加工处理。对于一些小型的图形、图标和按钮等,直接制作很方便,但这不足以描述自然景物和人像。也有一些较专业的绘画软件,通过数字化画板和画笔在屏幕上绘画。这种软件要求绘画者具有一定的美术知识及创意基础。

3. 利用数字转换设备采集图像

数字转换设备可以把采集到的图像转换成计算机能够记录和处理的数字图像数据。例如,对印刷品、照片或照相底片等进行扫描,用数码相机或数码摄像机对选定的景物进行拍摄等。从现实世界中获取数字图像所使用的设备通称为图像获取设备。一幅彩色图像可以视为二维连续函数 $f(x, y)$, 其彩色 f 是坐标 (x, y) 的函数。从二维连续函数到离散的矩阵表示,同样包含采样、量化和编码的数字化过程。数字转换设备获取图像的过程实质上是信号扫描和数字化的过程,它的处理步骤大体分为以下 3 步。

① 采样: 在 xy 坐标上对图像进行采样(也称为扫描)。如同声音信号在时间轴上的采样要确定采样频率一样,在图像信号坐标轴上的采样也要确定一个采样间隔,这个间隔即为图像分辨率。有了采样间隔,就可以逐行对原始图像进行扫描。首先设 y 坐标不变,对 x 轴按采样间隔得到一行离散的像素点 x_n 及相应的像素值。使 y 坐标也按采样间隔由小到大变化,就可以得到一个离散的像素矩阵 $[x_n, y_n]$, 每个像素点有一个对应的色

彩值。简单地说,将一幅画面划分为 $m \times n$ 个网格,每个网格称为一个取样点,用其亮度值来表示。这样,一幅连续的图像就转换为以取样点值组成的一个阵列(矩阵)。

② 量化:将扫描得到的离散像素点对应的连续色彩值进行 A/D 转换(量化),量化的等级参数即为图像深度。这样,像素矩阵中的每个点 (x_n, y_n) 都有对应的离散像素值 f_n 。

③ 编码:把离散的像素矩阵按一定方式编成二进制码组。最后,把得到的图像数据按某种图像格式记录在图像文件中。

6.3.4 图像的属性

描述一幅图像需要使用图像的属性。图像的属性包含分辨率、像素深度、真/伪彩色、图像的表示法和种类等。

1. 分辨率

人们常说的分辨率有两种,即显示分辨率和图像分辨率。

(1) 显示分辨率是指显示屏上能够显示出的像素数目。例如,显示分辨率 1024×768 表示显示屏分成 768 行(垂直分辨率),每行(水平分辨率)显示 1024 个像素,整个显示屏就含有 796 432 个显像点。屏幕能够显示的像素越多,说明显示设备的分辨率越高,显示的图像质量越高。

(2) 图像分辨率是指组成一幅图像的像素密度,也用水平和垂直的像素表示,即用每英寸多少点(dpi)表示数字化图像的大小。例如,用 200dpi 来扫描一幅 2×2.5 英寸的彩色照片,则会得到一幅 400×500 个像素点的图像。这实质上是数字化的采样间隔,从而确定组成一幅图像的像素数目。对同样大小的一幅图,如果组成该图像的像素数目越多,则说明图像的分辨率越高,图像看起来就越逼真。相反,图像显得越粗糙。因此,不同的分辨率会造成不同的图像清晰度。

图像分辨率与显示分辨率是两个不同的概念。图像分辨率确定的是组成一幅图像的像素数目,而显示分辨率确定的是显示图像的区域大小。它们之间的关系是:

(1) 图像分辨率大于显示分辨率时,在屏幕上只能显示部分图像。例如,当图像分辨率为 800×600 ,屏幕分辨率为 640×480 时,屏幕上只能显示一幅图像的 64% 左右。

(2) 图像分辨率小于屏幕分辨率时,图像只占屏幕的一部分。例如,当图像分辨率为 320×240 ,屏幕分辨率为 640×480 时,图像只占屏幕的四分之一。

2. 图像深度

图像深度是指存储每个像素所用的位数,也用于量度图像的色彩分辨率。图像深度确定彩色图像每个像素可能的颜色数,或者确定灰度图像每个像素可能的灰度级数。它决定了彩色图像中可出现的最多颜色数,或灰度图像中的最大灰度等级。如一幅图像的图像深度为 b 位,则该图像的最多颜色数或灰度级为 2^b 种。显然,表示一个像素颜色

的位数越多,它能表达的颜色数或灰度级就越多。例如,只有 1 个分量的单色图像,若每个像素有 8 位,则最大灰度数目为 $2^8=256$ 。一幅彩色图像的每个像素用 R、G、B 3 个分量表示,若 3 个分量的像素位数分别为 4、4、2,则最大颜色数目为 $2^{4+4+2}=2^{10}=1024$ 。也就是说像素的深度为 10 位,每个像素可以是 2^{10} 种颜色中的一种。表示一个像素的位数越多,它能表达的颜色数目就越多,它的深度就越深。

3. 真彩色和伪彩色

真彩色(true color)是指组成一幅彩色图像的每个像素值中,有 R、G、B 3 个基色分量,每个基色分量直接决定显示设备的基色强度,这样产生的彩色称为真彩色。例如,用 RGB8 : 8 : 8 方式表示一幅彩色图像,也就是 R、G、B 分量都用 8 位来表示,可生成的颜色数就是 2^{24} 种。每个像素的颜色就是由其中的数值直接决定的。这样得到色彩可以反映原图像的真实色彩,通常称作真彩色。在一些场合中,通常把 RGB8 : 8 : 8 方式表示的彩色图像称为真彩色图像或全彩色图像。

为了减少彩色图形的存储空间,在生成图像时,可对图像中的不同色彩进行采样,产生包含各种颜色的颜色表,即彩色查找表。图像中每个像素的颜色不是由三个基色分量的数值直接表达,而是把像素值作为地址索引,以便在彩色查找表中查找这个像素实际的 R、G、B 分量。人们将图像的这种颜色表达方式称为伪彩色。需要说明的是,这种伪彩色图像的数据除了保存代表像素颜色的索引数据外,还要保存一个色彩查找表(调色板)。彩色查找表可以是一个预先定义的表,也可以是对图像进行优化后产生的色彩表。常用的 256 色的彩色图像使用了 8 位的索引,即每个像素占用一个字节。

6.3.5 图形图像转换

图形和图像之间在一定的条件下可以转换,如采用光栅化(点阵化)技术可以将图形转换成图像。采用图形跟踪技术可以将图像转换成图形。一般可以通过硬件(输入输出设备)或软件实现图形和图像之间转换。

1. 图形和图像的硬件转换

一张工程图纸,一般认为它是图形,用扫描仪将它输入到 Photoshop,它就变成图像信息(点位图)。用数字化仪将它输入到 AutoCAD 后,它就变成图形信息(矢量图)。也就是说,同一个对象既可作为图形处理也可以作为图像处理。那么到底哪种过程更有效,这要看处理的对象性质和要达到的处理结果。当然,可以用扫描仪先将它扫进计算机,变成图像信息,再用一定的软件(如 Corel-Trace, Photoshop 的轮廓跟踪)人工或自动地勾勒它的轮廓,这个过程称为向量化,也就是图像转换为图形的过程。这个过程必然会丢失许多细节,所以通常仅适用于工程绘图领域。

如果我们用 AutoCAD 软件画好了一张图,较合理的方法是用绘图仪将它输出,当然

也可以用打印机将它输出。这时计算机必须先将图形转换为打印机的扫描线,这个过程称为光栅化,这就是图形转换为图像的过程。如果用 Photoshop 软件制作好了一张图,较合理的方法是用打印机将它输出,这样可以得到较多的层次和细节。如果一定要用绘图仪输出,就必然会丢失许多图像的细节。

2. 图形和图像的软件转换

图形和图像都是以文件形式存放在计算机存储器中的。可以通过应用软件实现文件格式之间的转换,达到图形和图像之间的转换。随着图形和图像处理技术的发展,出现了很多较好的格式转换软件,如 CorelDraw 软件,它几乎提供了所有常用图形图像文件格式之间的转化功能。同时也出现了一些较好的文件格式,如 Adobe 公司 EPS 格式文件,它是一种兼具图形图像各自优点的文件格式。转换并不表示可以任意互换,实际上许多转换是不可逆的。转换的次数越多,丢失的信息就越多,特别是图形和图像之间的转化。例如,当人们将一个 BMP 格式的文件转化为 GIF、TIFF 等格式的文件时问题还不大,但如果将它转化为 DIF 等格式的文件时就丢失了许多细节,甚至像一个矩形中间的填充色块,都用几条线表示。又如,将一个 AutoCAD 的 DWG 文件转化为 DIF 文件时问题还不大,但如果将它转化为 BMP、GIF 和 TIFF 文件时,就必须考虑分辨率和彩色数。这两个参数决定了最终图像文件的大小和它的使用价值。总之,从本质上讲,各种不同的文件格式是对不同性质的处理对象或同一对象的不同处理侧面采用一种最为科学、合理和方便的描述方法。应该根据处理对象的特点选择或转化为相应的文件格式,以及选择相应的输入输出设备。

6.3.6 图像的压缩编码

扫描生成一幅图像时,实际上就是按一定的图像分辨率和一定的图像深度对模拟图片或照片进行采样,从而生成一幅数字化的图像。图像分辨率越高,图像深度越深,则数字化后的图像效果越逼真,图像数据量越大。如果按照像素点及其深度映射的图像数据大小采样,可用下面的公式估算数据量:

$$\text{图像数据量} = \text{图像的总像素} \times \text{图像深度} / 8(B)$$

其中,图像的总像素为图像的水平方向像素数乘以垂直方向像素数。

例如,一幅 640×480 的 256 色图像,其文件大小约为:

$$640 \times 480 \times 8 / 8 \approx 300KB$$

可见,数字图像的数据量也很大,需要很大的存储空间存储数据图像。更重要的是,在现代通信中,特别是在互联网上开展的各种应用中,图像传输速度是一项很重要的指标。以使用拨号接入互联网的家庭用户为例,假设数据传输速度为 $56Kb/s$,则理想情况下,传输一幅分辨率为 640×480 的 6.5 万色的未经压缩的图像大约需要 1~2 分钟。因

此,采用压缩编码技术,减少图像的数据量,是提高网络传输速度的重要手段。

由于数字图像中数据的相关性很强,或者说,数据的冗余度很大,因此对数字图像进行大幅度的数据压缩是完全可能的。而且,人眼的视觉有一定的局限性,即使压缩前后的图像有一定失真,只要限制在人眼允许的误差范围之内,也是允许的。

数据压缩可分成两类,一类是无损压缩,另一类是有损压缩。无损压缩利用数据的统计冗余进行压缩,可以保证在数据的压缩和还原过程中,图像信息没有损耗或失真。图像还原(解压缩)时可完全恢复,即重建后的图像与原始图像完全相同。例如,在多媒体应用中,常用行程长度编码(RLE)、增量调制编码(DM)、霍夫曼(Huffman)编码以及LZW编码等。

1. 行程长度编码(run-length encoding, RLE)

某些图像往往有许多颜色相同的图块。在这些图块中,许多连续的扫描行都具有同一种颜色,或者同一扫描行上许多连续的像素都具有相同的颜色值。在这些情况下就不需要存储每一个像素的颜色值,而仅仅存储一个像素值以及具有相同颜色的像素数目。这种编码称为行程编码。具有同一颜色的连续像素的数目称为行程长度。其压缩率的大小取决于图像本身。如果图像中具有相同颜色的横向色块越大,图像块数目越多,压缩比就越大。反之就越小。

2. 增量调制编码(delta modulation encoding, DM)

在比较大的范围内,自然图像往往有颜色虽不完全一致,但变化不大的特点。因此,在这些区域中,相邻像素的像素值相差很小,具有很大的相关性。在一幅图像中,除了轮廓特别明显的地方以外,大部分区域都具有这种特点。增量调制编码就是利用图像相邻像素值的相关性来压缩每个像素值的位数,以达到减少存储容量的目的。增量调制编码压缩图像时,不存储扫描行上每个像素的实际值,仅存储每一行上第一个像素的实际值。其后,依次存储每一个像素的像素值与前一个像素值之差,即增量值。

3. 霍夫曼(Huffman)编码

大多数图像常常包含单色的大面积图块,而且某些颜色比其他颜色出现的更频繁,因此可以采用霍夫曼编码方式。霍夫曼编码的基本方法是先对图像数据扫描一遍,计算出各种像素出现的概率,按概率的大小指定不同长度的惟一码字,由此得到一张该图像的霍夫曼码表。编码后的图像数据记录的是每个像素的码字,而码字与实际像素值的对应关系记录在码表中。码表是附在图像文件中的。在实际应用中,霍夫曼编码常与其他编码方法结合使用,以获得更大的压缩比。

有损压缩方法利用人眼视觉对图像中的某些频率成分不敏感的特性,采用一些高效的有限失真数据压缩算法,允许压缩过程中损失一定的信息。采用有损压缩后的数据进行图像重建时,重建后的图像与原始图像虽有一定的误差,但并不影响人们对图像含义的

正确理解。却换来了较大的压缩比,大幅度减少了图像信息中的冗余信息。为了得到较高的数据压缩比,数字图像的压缩一般都采用有损压缩。经常使用的有损压缩方法有预测编码、变换编码、矢量编码和基于模型的编码。实际使用时常常是多种压缩方法的结合。

总之,图像压缩的方法很多,不同方法有不同的适用场合和范围。

6.3.7 多媒体数据压缩编码的国际标准

计算机中使用的图像压缩编码方法有多种国际标准和工业标准。目前广泛使用的编码及压缩标准有 JPEG、MPEG 和 H. 261。

JPEG(Joint Photographic Experts Group)是一个由 ISO 和 IEC 两个组织机构联合组成的专家组,负责制定静态和数字图像数据压缩编码标准。这个专家组地区性的算法称为 JPEG 算法,并且成为国际上通用的标准,因此又称为 JPEG 标准。JPEG 是一个适用范围很广的静态图像数据压缩标准,既可用于灰度图像又可用于彩色图像。JPEG 专家组开发了两种基本的压缩算法,一种是以离散余弦变换 DCT(discrete cosine transform,DCT)为基础的有损压缩算法,另一种是以预测技术为基础的无损压缩算法。使用有损压缩算法时,在压缩比为 25:1 的情况下,压缩后还原得到的图像与原始图像相比较,人们难于察觉它们之间的区别,因此得到了广泛的应用。例如,在 V-CD 和 DVD-Video 电视图像压缩技术中,就使用 JPEG 有损压缩算法来取消同方向上的冗余数据。为了保证图像质量的前提下进一步提高压缩比,JPEG 专家组制定了 ISO/IEC 15444 JPEG2000(简称 JP2000)标准,这个标准采用了小波变换(wavelet)算法。

动态图像压缩标准(Moving Pictures Experts Group,MPEG)是一个由 ISO 和 IEC 两个组织机构联合组成的活动图像专家组,于 1990 年形成了一个标准草案,将 MPEG 标准分成两个阶段:第一阶段(MPEG-1)是针对传输率为 1Mb/s 到 1.5Mb/s 的普通电视质量的视频信号的压缩;第二阶段(MPEG-2)目标则是对每秒 30 帧的 720×572 分辨率的视频信号进行压缩。在扩展模式下,MPEG-2 可以对分辨率达 1440×1152 高清晰度电视(HDTV)的信号进行压缩。MPEG 标准分成 MPEG 视频、MPEG 音频和视频音频同步 3 个部分。1999 年发布了 MPEG-4 多媒体应用标准,目前又推出了 MPEG-7 多媒体内容描述接口标准等。每个新标准的产生都极大地推动了数字视频的发展和更广泛的应用。

H. 261 视频通信编码标准是由国际电话电报咨询委员会(Consultative Committee on International Telephone and Telegraph,CCITT)于 1998 年提出的电话/会议电视的建议标准,该标准又称为 P×64K 标准,其中 P 是取值为 1~30 的可变参数。P=1 或 2 时支持四分之一通用中间格式(quarter common intermediate format,QCIF)的帧率较低

的视频电话传输。 $P \leq 6$ 时支持通向中间格式(common intermediate format, CIF)的帧率较高的电视会议数据传输。 $P \times 64K$ 视频压缩算法也是一种混合编码方案,即基于 DCT 的变换编码和带有运动预测差分脉冲编码调制(DPCM)的预测编码方法的混合。在低传输率($P=1$ 或 $P=2$, 即 64Kb/s 或 128Kb/s)时,除 QCIF 外还可以使用亚帧技术,即每间隔一帧(或数帧)处理一帧,压缩比例可达 50 : 1 左右。CCITT 推出的 H. 263 标准用于低传输速率通信的电视图像编码。

6.3.8 图形、图像文件格式

数字图像在计算机中存储时,其文件格式繁多。下面简单介绍几种常用的文件格式。

1. BMP 文件

BMP(bitmap-file) 图像文件是 Windows 操作系统采用的图像文件格式。在 Windows 环境下运行的所有图像处理软件几乎都支持 BMP 图像文件格式。它是一种与设备无关的位图格式,目的是为了让 Windows 能够在任何类型的显示设备上输出所存储的图像。BMP 采用位映射存储格式,除了图像深度可选以外,一般不采用其他任何压缩,所以占用的存储空间较大。BMP 文件的图像深度可选 1 位、4 位、8 位及 24 位,有黑白、16 色、256 色和真彩色之分。

2. GIF 文件

GIF 是 CompuServe 公司开发的图像文件格式,它以数据块为单位来存储图像的相关信息。GIF 文件格式采用了 LZW(Lempel-Ziv-Walch)无损压缩算法,按扫描行压缩图像数据。它可以在一个文件中存放多幅彩色图像,每一幅图像都由一个图像描述符、可选的局部彩色表和图像数据组成。如果把存储在一个文件中的多幅图像逐幅读出来显示到屏幕上,可以像播放幻灯片那样显示或者构成简单的动画效果。GIF 的图像深度为 1 位到 8 位,即最多支持 256 种色彩的图像。

GIF 文件格式定义了两种数据存储方式,一种是按行连续存储,存储顺序与显示器的显示顺序相同;另一种是按交叉方式存储。由于显示图像需要较长的时间,使用这种方法存放图像数据,用户可以在图像数据全部收到之前看到这幅图像的概貌,而不觉得等待时间太长。目前,GIF 文件格式在 HTML 文档中得到广泛使用。

3. TIFF 文件

TIFF(.TIF)文件是由 Aldus 和 Microsoft 公司为扫描仪和桌面出版系统研发的一种较为通用的图像文件格式。TIFF 是电子出版 CD-ROM 中一个重要的图像文件格式。TIFF 格式非常灵活易变,它又定义了 4 类不同的格式:TIFF-B 适用于二值图像;TIFF-G 适用于黑白灰度图像;TIFF-P 适用于带调色板的彩色图像;TIFF-R 适用于 RGB 真彩图像。无论在视觉上还是其他方面,都能把任何图像编码成二进制形式而不丢失任何属性。

4. PCX 文件

PCX 文件是 PC Paintbrush(PC 画笔)图像文件格式。PCX 的图像深度可选为 1 位、4 位或 8 位,对应单色、16 色及 256 色,不支持真彩色。PCX 文件采用 RLE 行程编码,文件体中存放的是压缩后的图像数据。因此,将采集到的图像数据写成 PCX 格式文件时,要对其进行 RLE 编码。而读取一个 PCX 文件时首先要对其进行解码,才能进一步显示和处理。

5. PNG 文件

PNG 文件是作为 GIF 的替代品而开发的,它能够避免使用 GIF 文件所遇到的许多常见问题。它从 GIF 那里继承了许多特征,增加了一些 GIF 文件所没有的特性。用来存储灰度图像时,灰度图像的深度可达 16 位。存储彩色图像时,彩色图像的深度可达 48 位。在压缩数据时,它采用了一种在 LZ77 算法基础上改进的无损压缩算法。

6. JPEG 文件

JPEG(.JPG)文件采用一种有损压缩算法,其压缩比约为 5:1 至 50:1,甚至更高。对一幅图像按 JPEG 格式进行压缩时,可以根据压缩比与压缩效果要求选择压缩质量因子。JPEG 格式文件的压缩比例很高,非常适用于处理大量图像的场所,它是一种有损压缩的静态图像文件存储格式,压缩比例可以选择。支持灰度图像、RGB 真彩色图像和 CMYK 真彩色图像。

7. Targe 文件

Targe(.TGA)文件格式用于存储彩色图像,可支持任意大小的图像,最高彩色数可达 32 位。专业图形用户经常使用 TGA 点阵格式保存具有真实感的三维有光源图像。

8. WMF 文件

WMF 文件只用于 Windows 中,它保存的不是点阵信息,而是函数调用信息。它将图像保存为一系列 GDI(图形设备接口)函数调用。在恢复时,应用程序执行源文件(即执行一个个函数调用),在输出设备上画出图像。WMF 文件具有设备无关性,文件结构好,但是解码复杂,效率比较低。

9. EPS 文件

EPS 文件是用 PostScript 语言描述的 ASCII 图形文件。在 PostScript 图形打印机上能打印出高品质的图形,能够表示 32 位图形(图像)。EPS 文件格式分为 Photoshop EPS 格式和标准 EPS 格式,其中标准 EPS 格式又可分为图形格式和图像格式。

10. DIF 文件

DIF 文件是 AutoCAD 中的图形,它以 ASCII 方式存储图像。表现图形在尺寸大小方面十分精确,可以被 CorelDraw 和 3DS 等软件调用编辑。

11. CDR 文件

CDR 文件是 CorelDraw 的文件格式,是所有 CorelDraw 应用程序均能使用的图形(图像)文件。

6.4 动画和视频

6.4.1 动画

动画是将静态的图像、图形及图画等按一定的时间顺序显示出来的,从而形成连续的动态画面。从传统意义上说,动画是通过在连续多格的胶片上拍摄一系列画面,并将胶片以一定的速度放映,从而产生动态视觉的技术和艺术。电影放映的标准是每秒放映 24 帧(画面),每秒遮挡 24 次,刷新率是每秒 48 次。一般说来,动画是一种动态生成一系列相关画面的处理方法,其中的每一幅与前一幅略有不同。计算机动画是在传统动画的基础上,使用计算机图形图像技术而迅速发展起来的一门高新技术。计算机动画采用连续播放静止图像的方法产生景物运动的效果,即使用计算机产生图形、图像运动的技术。计算机动画的原理与传统动画基本相同,只是在传统动画的基础上把计算机技术用于动画的处理和应用,并可以达到传统动画所达不到的效果。计算机的动画不仅实体在运动,而且色调、纹理、光影效果也可以不断改变。计算机生成的动画不仅可记录在胶片上,而且还可以记录在磁带、磁盘和光盘上。放映时不仅可以使使用计算机显示器显示,而且还可以使用电视机屏幕显示以及使用投影仪投影到银幕上的方法显示。在多媒体应用中,计算机动画可以十分简单,也可以十分复杂,从某个对象、物体的运动到电视广告、动画片等。

动画的本质是运动。根据运动的控制方式可将计算机动画分为实时动画和逐帧动画两种。实时动画是用算法来实现物体的运动的;逐帧动画是在传统动画基础上引申而来的,也即通过一帧一帧显示动画的图像序列而实现运动的效果。根据视觉空间的不同,计算机动画又有二维动画和三维动画之分。

1. 实时动画

实时动画采用各种算法来实现运动物体的运动控制。采用的算法有运动学算法、动力学算法、反向运动学算法、反向动力学算法以及随机运动算法等。在实时动画中,计算机对输入的数据进行快速处理,并在人眼察觉不到的时间内将结果随时显示出来。实时动画的响应时间与许多因素有关,如动画图像大小、动画图像复杂程度、运算速度快慢(计算机)以及图形的计算是采用软件还是硬件等。

2. 矢量动画

矢量动画是由矢量图衍生出的动画形式。矢量图是利用数学函数来记录和表示图形

线条、颜色、尺寸、坐标等属性的。矢量动画通过各种算法实现各种动画效果,如位移、变形和变色等。也就是说,矢量动画是通过计算机的处理,使矢量图产生运动效果而形成的动画。使用矢量动画,可以使一个物体在屏幕上运动,并改变其形状、大小、颜色、透明度、旋转角度以及其他一些属性参数。矢量动画能够采用实时绘制的方式显示一幅矢量图。当图形放大或缩小时,都保持光滑的线条,不会影响质量,也不会改变文件的容量。

3. 二维动画

二维动画是对传统动画的一个改进,它不仅具有模拟传统动画制作的功能,而且可以发挥计算机所特有的功能,如生成的图像可以复制、粘贴、翻转、放大缩小、任意移位以及自动计算等。图形、图像技术都是计算机动画处理的基础。图像是指用像素点组成的画面,而图形是指由几何形体组成的画面。在二维动画处理中,图像技术有利于绘制实际景物,可用于绘制关键帧、画面叠加以及数据生成。图形技术有利于处理线条组成的画面,可用于自动或半自动中间画面的生成。计算机在二维动画中的作用包括:输入和编辑关键帧;计算和生成中间帧;定义和显示运动路径;产生特技效果;实现画面与声音的同步;控制运动系列的记录等。二维动画的处理主要包括两个基本步骤:第一个步骤是屏幕绘画;第二个步骤是动画生成。屏幕绘画主要利用图像处理软件完成。为了自动或半自动生成动画,有时也采用图形方法来描述画面。动画生成以屏幕绘画的结果(作为关键帧)为基础进行生成处理,最终完成动画创作。

4. 三维动画

三维画面中的景物有正面,也有侧面和反面。调整三维空间的视点,能看到不同的内容。二维画面则不然,无论怎么看,画面的内容都是不变的。三维与二维动画的区别主要在于采用不同的方法获得动画中的景物运动效果。三维动画的制作过程不同于传统动画制作。根据剧情的要求,首先要建立角色、实物和景物的三维数据模型,再对模型进行光照射色(真实感设计),然后使模型动起来。即模型可以在计算机控制下在三维空间中运动,或近或远,或旋转或移动,或变形或变色等,最后对运动的模型重新生成图像并刷新屏幕,形成运动图像。

建立三维动画物体模型称为造型,也就是在计算机内生成一个具有一定形体的几何模型。计算机中大致有3种形式来记录一个物体的模型。

(1) 线框模型:用线条来描述一个形体,一般包括顶点和棱边。例如用八条线来描述一个立方体。

(2) 表面模型:用面的组合来描述形体,如用六个面来描述一个立方体。

(3) 实体模型:任何一个物体都可以分解成若干个基本形体的组合,如一个立方体可以分解为各种形体的组合。这种用基本形体组合物体的模型就是实体模型。

三维动画的处理需要综合使用上述3种模型。一般情况下,先用线框模型进行概念

设计,再将线框模型处理成表面模型以方便显示,使用实体模型进行动画处理。同一形体的3种模型可以相互转换。

物体模型只有通过光和色的渲染,才能产生自然界中常见的真实物体效果,这在动画中称为着色(真实感设计)。对物体着色是产生真实感图形图像的重要过程,它涉及到物体的材质、纹理以及照射的光源等方面。

(1) 材质:除了造型以外,描述任何物体都必须有一定的附加特征(属性)来指明它的外在特性。物体的外在特性在很大程度上取决于构成它的材料。一般把材料的性质简称为材质。不同的材质表现出的质感是不同的。材质主要用来说明物体对入射光线做出的反应。一般来说,光线照射到物体后,或者被反射、吸收,或者被透射、折射。反射的色光正是该物体呈现的颜色,而透射、折射产生的色光与材质有很大关系。

(2) 纹理:纹理是物体表面细节,大多数物体的表面具有纹理。有了纹理可以改变物体的外观,甚至改变其形状。物体的纹理一般分为两种:一种是颜色纹理,如墙面贴纸和陶器上的图案等,颜色纹理取决于物体表面的光学性质;另一种是几何纹理,如人的皮肤和橘子的褶皱等。几何纹理与物体表面的微观几何形状有关。

(3) 光源:给一个场景着色时必须知道有关光源的特性:光源的位置、颜色、亮度和方向等,这些信息要由用户通过照明模型设定。决定是否有光照射到物体表面并形成颜色的方法称为浓淡处理。浓淡处理要用到物体表面的几何材质信息,并对入射光进行考察,从而找出表面反射的色光。在进行浓淡处理时,一般对每个物体表面上的每一个点分别进行处理,而且仅考虑来自光源的光照效果,而不考虑来自其他物体发出的光的影响。

三维动画处理的基本目的是控制形体模型的运动,获得运动显示效果。其处理过程中涉及建立线框模型、表面模型和实体模型。此外,一个好的三维动画应用系统能够将形体置于指定的灯光环境中,使形体的色彩在灯光下生成光线反映和阴影效果。运动物体不仅表现为几何位置改变,还带有光、色、受力、碰撞以及物体本身的变形等。动画控制也称为运动模拟。首先,计算机要确定每个物体的位置和相互关系,建立其运动轨迹和速度,选择运动形式(平移、旋转和扭曲等)。然后,须确定物体形体的变形方式和变异速度。如果光源确定好了以后,调整拍摄的位置、方向、运动轨迹及速度,就可以显示,观看画面效果。

三维动画最终要生成一幅幅二维画面,并按一定格式记录下来,这个过程称为动画生成。动画生成后,可以在屏幕上播放,也可以录制在光盘或录像带上。

6.4.2 模拟视频

1. 模拟视频原理

电视是当代最有影响的多媒体信息传播工具。在综合文、图、声及像等作为信息传播媒体这一点上完全与多媒体系统相同。不同的是电视系统不具备交互性,传播的信号是

模拟信号。电视信号记录的是连续的图像或视像及伴音(声音)信号。电视信号通过光栅扫描的方法显示在荧光屏(屏幕)上。扫描从荧光屏的顶部开始,一行一行地向下扫描,直至荧光屏的最底部,然后返回到顶部,重新开始扫描。这个过程产生的一个有序的图像信号的集合,组成了电视图像中的一幅图像,称为一帧。连续不断的图像序列就形成了动态视频图像。水平扫描线所能分辨出的点数称为水平分辨率,一帧中垂直扫描的行数称为垂直分辨率。一般来说,点越小,线越细,分辨率越高。每秒钟所扫描的帧数称作帧频,一般在每秒 25 帧时人眼就不会感觉到闪烁。彩色电视系统采用相加混色,使用 RGB 作为 3 基色进行配色,产生 R、G、B 3 个输出信号。RGB 信号可以分别传输,也可以组合起来传输。根据亮度色度原理,任何彩色信号都可以分解为亮度和色度。

2. 彩色电视的制式

电视信号的标准也称为电视的制式,目前世界各地使用的标准不完全相同,制式的区分主要在于其帧频的不同,分辨率的不同,信号带宽及载频的不同,以及彩色空间的转换关系不同等。世界上现行的彩色电视制式主要有 NTSCM 制、PAL 制和 SECAM 制 3 种,如表 6-2 所示。

美国、加拿大、日本、韩国、中国台湾和菲律宾等国家和地区采用 NTSCM 制式。德国、英国、中国、中国香港和新西兰等国家和地区采用 PAL 制式。法国、东欧和中东一带采用 SECAM 制式。

表 6-2 彩色数字电视制式

TV 制式	帧频 (Hz)	行/帧	亮度带宽 (MHz)	彩色副载波 (MHz)	色度带宽 (MHz)	声音载波 (MHz)
NTSCM	30	525	4.2	3.58	1.3(I)、0.6(Q)	4.5
PAL	25	625	6.0	4.43	1.3(U)、1.3(V)	6.5
SECAM	25	625	6.0	4.25	>1.0(U)、>1.0(V)	6.5

我国电视制式(PAL)采用 625 行隔行扫描光栅,分两场扫描。行扫描频率为 15625Hz,周期为 64 μ s。场扫描频率为 50Hz,周期为 20ms。帧频是 25Hz,周期为 40ms。在发送电视信号时,每一行中传送图像的时间是 52.2 μ s,对应行扫描的正程时间,其余的 11.8 μ s 不传送图像。对应行扫描的逆程时间加入行消隐信号和行同步信号,将不影响行扫描发送或显示图像信息。每一场扫描的行数为 625/2 行,其中 25 行作场回扫,不传送图像。

采用隔行扫描比采用逐行扫描所占用的信号传输带宽要减少一半,有利于信道的利用,有利于信号传输和处理。采用每秒 25 帧的帧频(25Hz)能以最少的信号容量有效地满足人眼的视觉残留特性。采用 50Hz 的场频是因为我国的电网频率为 50Hz,采用

50Hz 的场刷新频率可以有效地去掉电网信号的干扰。

6.4.3 数字视频

视频信息是指活动的、连续的图像序列。一幅图像称为一帧,帧是构成视频信息的基本单元。在多媒体应用系统中,视频以其直观和生动等特点得到广泛的应用。视频与动画一样,是由一幅幅帧序列组成的。这些帧以一定的速率播放,使观看者得到连续运动的感觉。计算机的数字视频是基于数字技术的图像显示标准,它能将模拟视频信号输入到计算机进行数字化视频编辑,从而制成数字视频。全屏幕视频是指显示的视频图像充满整个屏幕,能以 30 帧/秒的速度刷新画面,使画面不会产生闪烁和不连贯的现象。电视机、激光视盘和摄像机等都可提供丰富多彩的模拟视频信号。常常需要把这些信号与计算机图形图像结合在一个共同的空间,通过处理达到最佳的效果,然后输出到计算机的显示器或其他电视设备上。模拟视频信号进入计算机,首先需要解决模拟视频信息的数字化问题。与音频数字化一样,视频数字化的目的是将模拟信号经模数转换和彩色空间变换等过程,转换成计算机可以显示和处理的数字信号。由于电视和计算机的显示机制不同,因此要在计算机上显示视频图像需要做许多处理。例如,电视是隔行扫描的,而计算机的显示器通常是逐行扫描。电视是亮度(Y)和色度(C)的复合编码,而 PC 机的显示器工作在 RGB 空间。电视图像的分辨率和显示屏的分辨率也各不相同。这些问题在电视图像数字化过程中都需要考虑。通常,对模拟视频信息进行数字化应采取如下方式:

(1) 先从复合彩色电视图像中分离出彩色分量,然后数字化。目前市场上的大多数电视信号都是复合的全电视信号,如录像带、激光视盘等存储设备上的电视信号。对这类信号的数字化,通常是将其分离成 YUV、YIQ 或 RGB 彩色空间的分量信号,然后用 3 个 A/D 转换器分别进行数字化。这种方式称为复合数字化。

(2) 先对全彩色电视信号数字化,然后在数字域中进行分离,以获得 YUV、YIQ 或 RGB 分量信号。用这种方法对电视图像数字化时,只需一个高速 A/D 转换器。这种方式称为分量数字化。

视频信息数字化的过程比声音复杂一些,它是以一幅幅彩色画面为单位进行的。分量数字化是较多使用的一种方式。电视信号使用的彩色空间是 YUV 空间,即每幅彩色画面有亮度(Y)和色度(U、V)3 个分量。对这 3 个分量须分别进行取样和量化,得到一幅数字图像。由于人眼对色度信号的敏感程度远不如对亮度信号那么灵敏,所以色度信号的取样频率可以比亮度信号的取样频率低一些,以减少数字视频的数据量。目前使用的色度信号取样格式如表 6-3 所示。

表 6-3 色度信号取样格式

格 式	说 明
4:4:4 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y, 4 个色度样本 C_r 和 4 个色度样本 C_b , 这相当于每个像素用 3 个样本表示
4:2:2 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y, 两个色度样本 C_r 和 2 个色度样本 C_b , 平均每个像素用两个样本表示
4:2:0 格式	指在水平和垂直方向上每两个连续的取样点上取两个亮点样本 Y, 1 个色度样本 C_r 和 1 个色度样本 C_b , 平均每个像素用 1.5 个样本表示。H. 261、H. 263 和 MPEG-1 视频标准均使用这种取样格式。每个像素平均使用 12b 来表示
4:1:1 格式	指在每条扫描线上每 4 个连续的取样点取 4 个亮点样本 Y, 1 个色度样本 C_r 和 1 个色度样本 C_b , 平均每个像素用 1.5 个样本表示。数字录像机 DVC 使用这种格式

CCIR601 标准推荐使用 4:2:2 格式, 并对采样频率、采样结构以及彩色空间转换等都做了严格的规定。使用 4:2:2 格式时, 亮点样本 Y 用 13.5MHz 采样频率, 色度样本 C_r 、 C_b 用 6.75MHz 的采样频率。

6.4.4 数字视频标准

国际无线电咨询委员会(International Radio Consultative Committee, CCIR)制定的广播级质量数字电视编码标准, 即 CCIR 601 标准, 为 PAL、NTSC 和 SECAM 电视制式之间确定了共同的数字化参数。该标准规定了彩色电视图像转换成数字图像所使用的采样频率、采样结构以及彩色空间转换等。这个标准对多媒体的开发和应用十分重要。

1. 采样频率

CCIR 为 PAL、NTSC 和 SECAM 电视制式制定的共同的电视图像采样频率标准为:
采样频率=13.5MHz

2. 分辨率

PAL 和 SECAM 制式的亮度信号, 每一扫描行采集 864 个样本点。而对于 NTSC 制式的亮度信号, 每一扫描行采集 858 个样本点。CCIR 601 规定对所有制式, 每一扫描行的有效样本点数均为 720 个。

3. 数据量

CCIR 601 规定, 每个样本点都按 8 位数字化, 即有 256 个等级。但实际上亮度信号占 220 级, 色度信号占 225 级, 其他位作同步和编码等控制使用。

6.4.5 视频压缩编码

数字图像数据的数据量大, 而数字视频信息的数据量就更加突出。例如, 每帧

352×240 像素点,图像深度 16 位的图像,其数据量约为 1.3Mb,每秒 30 帧,其数据量就高达 40Mb/s。这样大的数据量无论是传输、存储还是处理,都是极大的负担。为了解决这个问题,必须对数字视频信息进行压缩编码处理。

视频压缩的目标是在尽可能保证视觉效果的前提下减少视频数据率。视频是连续的静态图像,其压缩编码算法与静态图像的压缩编码算法有某些共同之处。但视频还有其自身的特性,在压缩时必须考虑其运动特性。由于视频信息中各画面内部有很强的信息相关性,相邻画面又有高度的相容性(连贯性),再加上人眼的视觉特性,所以数字视频的数据量可压缩几十倍甚至几百倍。视频信息压缩编码的方法很多,在选择或设计视频压缩编码算法时需要掌握一些视频压缩的基本概念。

1. 无损压缩与有损压缩

视频压缩中的无损和有损压缩概念与静态图像基本类似。无损压缩指压缩前和解压后的数据完全一致。多数的无损压缩都采用 RLE 行程编码算法。这种算法特别适合于由计算机生成的图像,它们一般具有连续的色调。但是无损算法一般对数字视频和自然图像的压缩效果不理想,因为其色调细腻,不具备大块连续色调。

有损压缩意味着解压缩后的数据与压缩前的数据不一致。在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或声音信息,而且丢失的信息不可恢复。几乎所有高压压缩的算法都采用有损压缩,这样才能达到低数据率的目标。丢失的数据率与压缩比有关,压缩比越小,丢失的数据越多,解压缩后的效果越差。此外,某些有损压缩算法采用多次重复压缩的方式,这样还会引起额外的数据丢失。

2. 帧内和帧间压缩

帧内压缩也称为空间压缩。同一景物表面上各采样点的颜色之间往往存在着连贯性,但基于离散像素采样表示景物颜色的方式通常没有利用景物表面颜色的空间连贯性,从而产生了空间冗余。当压缩一帧视频时,仅考虑本帧的数据而不考虑相邻帧之间的冗余信息,这实际上与静态图像压缩类似。由于帧内压缩时各个帧之间没有考虑相互关系,所以压缩后的视频数据仍可以以帧为单位进行编码。帧内压缩一般达不到很高的压缩效果(很小的压缩比)。

视频具有运动的特性,故还可以采用帧间压缩的方法。采用帧间压缩是由于许多视频或连续的动画前后两帧具有很大的相关性,或者说前后两帧信息变化很小的特点。例如,在演示一个球在静态背景前滚动的视频片段中,连续两帧中的大部分图像基本是不变的(背景不变),即连续的视频其相邻帧之间具有冗余信息。根据这一特性,压缩相邻之间的冗余就可以进一步提高压缩量。帧间压缩也称为时间压缩,它通过比较时间轴上不同帧之间的数据进行压缩。帧差值算法是一种典型的时间压缩法,它通过比较本帧与相邻帧之间的差异,仅记录本帧与其相邻帧的差值,这样可以大大减少数据量。例如,如果一

段视频不包含大量超常的剧烈运动景象,而是由一帧一帧的正常运动构成的,采用这种算法就可以达到很好的效果。

3. 对称和不对称编码

对称性是压缩编码的一个关键特征。对称编码意味着压缩和解压缩占用相同的计算处理能力和时间。对称算法适合实时压缩和传送视频,如视频会议应用就以采用对称压缩编码算法为好。而在电子出版和其他多媒体应用中,一般把视频预先压缩处理好,尔后再播放,因此可以采用不对称编码。不对称编码意味着压缩时需要花费大量的处理能力和时间,而解压缩时则能较好地实时回放,即以不同的速度进行压缩和解压缩。一般地说,压缩一段视频的时间比回放(解压缩)该视频的时间要多得多。例如,压缩 3 分钟的视频片段可能需要 10 多分钟的时间,而该片段实时回放只需 3 分钟。

目前,国际标准化组织制定的有关视频(及其伴音)压缩编码的几种标准及其应用范围可参见表 6-4。

表 6-4 压缩编码的标准

名 称	源图像格式	压缩后的码率	主要应用
MPEG-1	CIF 格式	1.5Mb/s	适用于 VCD
CCITT H. 261	CIF 格式	$P \times 64 \text{ Kb/s}$	应用于视频通信,如可视电话、会议电视等
	QCIF 格式	$P=1,2$ 时支持 QCIF $P \geq 6$ 时支持 QIF	
MPEG-2	$720 \times 576 \times 25$	5~15Mb/s	DVD、150 路卫星电视直播等
	$352 \times 288 \times 25$	$< 5 \text{ Mb/s}$	交互式多媒体应用等
	$1440 \times 1152 \times 50$	80Mb/s	HDTV 领域
MPEG-4	多种不同的视频格式	最低可达 64Kb/s	虚拟现实、远程教育及交互式视频等

6.4.6 视频文件格式

1. GIF 文件

GIF(graphics interchange format)是 CompuServe 公司推出的一种高压缩比的彩色图像文件。GIF 格式采用无损压缩方法中效率较高的 LZW 算法,主要用于图像文件的网络传输。考虑到网络传输的实际情况,GIF 图像格式除了一般的逐行显示方式之外,还增加了渐显方式,也就是说,在图像传输过程中,用户可以先看到图像的大致轮廓。然后随着传输过程的继续而逐渐看清图像的细节部分,从而适应用户的观赏心理。目前因特网上大量采用的彩色动画文件多为这种 GIF 格式。

2. Flic 文件

Flic(.FLI/.FLC)文件是 Autodesk 公司在其出品的 Autodesk Animator/ Animator

Pro/3D Studio 等 2D/3D 动画制作软件中采用的彩色动画文件格式。其中, FLI 是最初基于 320×200 分辨率的动画文件格式, . FLC 是 FLI 的进一步扩展, 采用了更高效的数据压缩技术, 其分辨率也不再局限于 320×200 。Flic 文件采用行程编码(RLE)算法和 Delta 算法进行无损数据压缩, 具有较高的数据压缩率。

3. AVI 文件

AVI(audio video interleaved)是 Microsoft 公司开发的一种符合 RIFF 文件规范的数字音频与视频文件格式, Windows 95/98、OS/2 等多数操作系统直接支持。AVI 格式允许视频和音频交错在一起同步播放, 支持 256 色和 RLE 压缩, 但 AVI 文件并未限定压缩标准。因此, AVI 文件格式只是作为控制界面上的标准, 不具有兼容性。用不同压缩算法生成的 AVI 文件, 必须使用相同的解压缩算法才能播放出来。AVI 文件目前主要应用多媒体光盘上, 用来保存电影、电视等各种影像信息。有时也出现在因特网上, 供用户下载、欣赏新影片的片段。

4. Quick Time 文件

Quick Time(. MOV/. QT)是 Apple 公司开发的一种音频、视频文件格式, 用于保存音频和视频信息, 具有先进的视频和音频功能, 被 Apple Mac OS、Windows 95/98/NT 等主流平台支持。Quick Time 文件支持 25 位彩色, 支持 RLE、JPEG 等领先的集成压缩技术, 提供 150 多种视频效果, 并配有提供了 200 多种 MIDI 兼容音响和设备的声音装置。新版本的 Quick Time 进一步扩展了原有功能, 包含了基于 Internet 应用的关键特性, 能够通过 Internet 提供实时的数字化信息流、工作流与文件回放功能。此外, Quick Time 还采用了 Quick Time VR(QTVR)技术的虚拟现实技术, 用户通过鼠标或键盘的交互式控制, 可以观察某一地点周围 360 度的景象, 或者从空间任何角度观察某一物体。Quick Time 以其领先的多媒体技术和跨平台特性, 较小的存储空间要求, 技术细节的独立性以及系统的高度开放性, 得到广泛的认可和应用。

5. MPEG 文件

MPEG(. MPEG/. MPG/. DAT)文件格式是运动图像压缩算法的国际标准, 它包括 MPEG 视频、MPEG 音频和 MPEG 系统(视频、音频同步)3 个部分。MPEG 压缩标准是针对运动图像设计的, 其基本方法是: 单位时间内采集并保存第一帧信息, 然后只存储其余帧对第一帧发生变化的部分, 从而达到压缩的目的。MPEG 的平均压缩比为 $50:1$, 最高可达 $200:1$, 压缩效率非常高, 同时图像和音响的质量也非常好, 并且在 PC 机上有统一的标准格式, 兼容性相当好。

6. RealVideo 文件

RealVideo(. RM)文件是 Real Networks 公司开发的一种新型流式视频文件格式, 它包含在 Real Networks 公司制定的音频视频压缩规范 RealMideo 中, 主要用来在低速率

的广域网上实时传输活动视频影像。可以根据网络数据传输速率的不同而采用不同的压缩比率,从而实现影像数据的实时传输和实时播放。RealVideo 除了可以以普通的视频文件形式播放之外,还可以与 RealVideo 服务器相配合,在数据传输过程中边下载边播放视频影像,而不必像大多数视频文件那样,必须下载后才能播放。

6.5 多媒体网络

计算机网络是将多个计算机连接起来以实现计算机通信及多媒体信息共享。多媒体空间的合理分布和有效的协同操作将极大地缩小个体与群体、局部与全球的工作差距。超越时空限制,充分利用信息,协同合作,相互交流,节约大量的时间和经费等是多媒体的基本目标。

网络具备传播信息的强大功能,并且在实际生活中扮演了媒体的角色。其次,国内各种机构都已开辟网上传播的新领域,网络报纸杂志、网络广播娱乐、网上教育以及电子商务等业务应运而生,互联网逐渐成为大众传播的媒介。

6.5.1 超文本与超媒体

1. 超文本的概念

超文本是一种文本,与一般文本文件的差别主要是组织方式不同。它将文本中遇到的一些相关内容通过链接组织在一起,用户可以很方便地阅览这些相关内容。超文本是一种文本管理技术,它以节点为单位组织信息,在节点与节点之间通过表示它们之间关系的链加以连接,构成特定内容的信息网络。节点、链和网络是超文本所包含的 3 个基本要素。

(1) 节点:超文本中存储信息的单元,由若干个文本信息块(可以是若干屏、窗口、文件或小块信息)组成。节点大小按需要而定。

(2) 链:建立不同节点(信息块)之间的联系。每个节点都有若干个指向其他节点或从其他节点指向该节点的指针,这种指针称为链。链通常是有向的,即从链源(源节点)指向链宿(目的节点)。链源可以是热字、热区、图元、热点或节点等。一般链宿都是节点。

(3) 网络:由节点和链组成的一个非单一、非顺序的非线性网状结构。

文本中的词、短语、图像、声音剪辑或影视剪辑之间的链接,或者其他文件、超文本文件的链接,称为超链接(热链接)。词、短语、图像、声音剪辑或影视剪辑和其他文件通常被称为对象或者称为文档元素,因此超链接是对象之间或者文档元素之间的链接。建立相互链接的对象不受空间位置的限制,它们可以在同一个文件内也可以在不同的文件之间,还可以通过因特网与世界上的任何一台联网计算机上的文件建立链接关系。

2. 超媒体的概念

用超文本方式组织和处理的多媒体信息就是超媒体。超媒体不仅包含文字,而且还可以包含图形、图像、动画、声音和影视图像片段,这些媒体之间也是用超链接组织的,而且它们之间的链接也是错综复杂的。超媒体与超文本之间的不同是,超文本主要以文字的形式表示信息,建立的链接关系主要是文句之间的链接关系。超媒体除使用文本外,还使用图形、图像、动画、声音或影视片段等多种媒体来表示信息,建立的链接关系是图形、图像、动画、声音或影视片段等多种媒体之间的链接关系。

网页是 Web 站点上的文档。在 Web 网页上,为了区分有链接关系和没有链接关系的文档元素,有链接关系的文档元素通常都用不同颜色或下划线来表示。

6.5.2 流媒体的基本概念

流媒体是指在网络中使用流式传输技术的连续时基媒体,而流媒体技术是指把连续的影像和声音信息经过压缩处理之后放到专用的流服务器上,让浏览者一边下载一边观看或收听,而不需要等到整个多媒体文件下载完成的技术。流媒体融合了多种网络以及音视频技术,在网络中实现流媒体技术,必须完成流媒体的制作、发布、传播以及播放等环节。

(1) 流媒体系统通过某种流媒体技术,完成流媒体文件的压缩生成,经过服务器发布,然后在客户端完成流媒体文件解压播放的整个过程。因此,一个流媒体系统一般由三部分组成:流媒体开发工具,用来生成流式格式的媒体文件;流媒体服务器组件,用来通过网络服务器发布流媒体文件;流媒体播放器,用于客户端对流媒体文件的解压和播放。目前应用比较广泛的流媒体系统主要有 Windows Media 系统、Real System 系统和 Quick Time 系统等。

(2) 流媒体的传输一般采用建立在用户数据报协议 UDP(user datagram protocol)上的实时传输协议和实时流协议 RTP/RTSP 来传输实时的影音数据。RTP 是针对多媒体数据流的一种传输协议,它被定义为在一对一或一对多的传输情况下工作,提供时间信息,实现流同步。RTSP 协议定义了一对多的应用程序如何有效地通过 IP 网络传送多媒体数据。

(3) 流式文件格式与多媒体压缩文件有所不同,编码的目的是为了适合在网络环境中边下载边播放。将压缩文件编码成流式文件,还须增加许多附加信息,以便使客户端接收到的数据包可以重新有序地播放。实际的网络应用环境中并不包含流媒体数据文件,而是流媒体发布文件,例如, RAM 和 ASX 等,它们本身不提供压缩格式,也不描述影视数据,其作用是以特定的方式安排影视数据的播放。不同流媒体系统具有不同的流式文件格式,例如,Real System 系统支持的文件格式有 RM、RA、RP 和 RT。Windows

Media 系统支持的文件格式是 ASF 和 ASX。

(4) 浏览器通过互联网邮件 MIME 来识别各种不同的简单文件格式。Web 浏览器都是基于 HTTP 协议的,HTTP 协议中都内建有 MIME。Web 浏览器能够通过 HTTP 中内建的 MIME 来标记 Web 上的多媒体文件格式,包括各种流式文件。

(5) 媒体播放器是一个应用软件,主要功能是用于播放多种格式的音频、视频序列。可以作为单独的应用程序运行,或作为复合文档中的一个嵌入对象。

6.5.3 互联网上获取声音和影视的方法

多媒体网络技术(multimedia networking)在互联网上有很多应用,大致可分成两类:一类是以文本为主的数据通信,包括文件传输、电子邮件、网络新闻和 Web 等;另一类是以声音和视频图像为主的通信,通常把任何一种声音通信或图像通信的网络应用称为多媒体网络应用。

通常,声音或视频文件放在 Web 服务器上,由 Web 服务器通过 HTTP 协议把文件传送给用户。也可将声音或视频文件放在声音/视频流式播放服务器(streaming server)上,由流式播放服务器通过流式播放协议把文件传送给用户。流式播放服务器简称流式服务器。

目前声音和视频点播应用还没有完全直接集成到现在的 Web 浏览器中,所以一般采用媒体播放器来播放声音、音乐、动画和影视。典型的媒体播放器具有解压缩、消除抖动、纠错以及用户播放控制等功能。现在可以将多媒体应用插件(Plug in)嵌入浏览器内部,与浏览器软件协同工作。这种技术把媒体播放器的用户接口软件放在 Web 客户机的用户界面上,浏览器在当前 Web 页面上为其保留屏幕空间,并且由媒体播放器来管理。客户机可使用多种方法来读取声音和影视文件,其中常见的方法有如下 3 种。

1. 通过 Web 浏览器把声音/影视文件从 Web 服务器传送给媒体播放器

客户机读取声音/影视文件最简方法是将声音/影视文件放到 Web 服务器上,然后通过浏览器把文件传送给媒体播放器,其过程如下:

(1) Web 浏览器与 Web 服务器建立 TCP 连接,然后提交 HTTP 请求,请求传送声音/影视文件。

(2) Web 服务器向 Web 浏览器发送请求响应消息以及请求的声音/影视文件。

(3) Web 浏览器检查 HTTP 响应消息中的内容的类型,调用相应的媒体播放器,然后把声音/影视文件或者是指向文件的指针传递给媒体播放器。

(4) 媒体播放器播放声音/影视文件。

采用这种方法时,媒体播放器必须通过 Web 浏览器才能从 Web 服务器上得到声音/影视文件,需要把整个文件下载到浏览器之后再传送给媒体播放器。这样就会产生播放

延迟,影响播放。因此,存在较大的延迟问题。

2. 直接把声音/影视文件从 Web 服务器传送给媒体播放器

采用媒体播放器与 Web 服务器直接建立链接的方法,可以改进通过 Web 浏览器产生的延迟。在 Web 服务器和媒体播放器之间建立直接的 TCP 连接,可以把声音/影视文件直接传送给媒体播放器,其实现方法如下:

(1) 用户通过超级链接以请求传送声音/影视文件。

(2) 这个超级链接不是直接指向声音/影视文件,而是指向一个播放说明文件,这个文件含有实际的声音/影视文件的地址(URL)。播放说明文件被封装在 HTTP 消息中。

(3) Web 浏览器接收 HTTP 响应消息中的内容的类型,调用相应的媒体播放器,然后把响应消息中的播放说明文件传送给媒体播放器。

(4) 媒体播放器直接与 Web 服务器建立 TCP 连接,然后把传送声音/影视文件的 HTTP 请求消息发送到 TCP 连接上。

(5) 在 HTTP 响应消息中把声音/影视文件传送给媒体播放器并开始播放。

这种方法依然使用 HTTP 传送文件,不容易获得满意的交互性能(用户与 Web 服务器),如暂停、从头开始播放等。

3. 通过多媒体流放服务器将声音/影视文件传送给媒体播放器

这是一种采用 Web 服务器和流放服务器,将声音/影视文件直接传送给媒体播放器的方法。Web 服务器用于 Web 页面服务,流放服务器用于声音/影视文件服务。其结构参见图 6-2。采用这种结构,媒体播放器向流放服务器请求传送文件,而不是向 Web 服务器请求传送文件。媒体播放器和流放服务器之间使用流式播放协议进行通信,声音/影视文件可以使用 UDP(用户数据报协议)直接从流放服务器传送给媒体播放器。

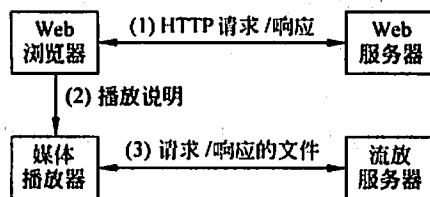


图 6-2 声音/影视文件传送给媒体播放器

6.6 多媒体计算机系统

一般的多媒体系统是由多媒体硬件系统和多媒体软件系统组成的。也即将多媒体信息和计算机交互式控制相结合,由对媒体信号的获取、生成、存储、处理和传输等数字化技

术所组成的一个完整的系统。

通常将具有多种媒体处理能力的计算机称为多媒体计算机(Multimedia Personal Computer, MPC)。传统的 PC 机处理的信息往往仅限于文字和数字,人机之间的交互只能通过键盘和显示器,缺乏多样性的信息交流途径。为了改变人机交互的接口,使计算机能够集声、文、图和像处理于一体,适应多媒体系统功能目标和应用需求,人们一方面改进 PC 机体系结构,使 PC 机性能升级,适应更丰富、更复杂的数据类型。芯片设计技术的发展,已将多媒体和通信功能集成到了 CPU 芯片中,形成了专用的多媒体微处理器,使得处理音频和视频就如处理数字和文字一样快捷。为了加快多媒体信息处理的速度, Micro Unity、Philips 等公司将媒体处理器与通用的 CPU 结合,扩展了 CPU 的多媒体处理和通信功能。Intel 公司推出了带有 MMX 技术的处理器。MMX 技术提供了面向多媒体和通信功能的特性,并保持了微处理器的体系结构。另一方面运用多媒体专用芯片和板卡,集成以 PC 机为中心的组平台。随着微电子集成电路技术和计算机技术的发展,现代高性能 PC 机为适应各种应用领域对处理速度和容量的要求,其体系结构发生了很大的变化,性能得到很大的提高,可以构成多媒体计算机。目前,PC 机的多媒体功能大都通过附加插件和设备实现,如音频卡、视频卡、3D 图形卡、网络卡以及 CD-ROM 驱动器、扫描仪和数码相机等。因此,一个完整的多媒体计算机系统是由多媒体计算机硬件和多媒体计算机软件组成的。

6.6.1 多媒体计算机硬件系统

多媒体硬件系统可以看作在 PC 机的基础上进行了硬件扩充,以适应多媒体信息处理的功能需求。计算机硬件及声像等媒体输入输出设备构成了多媒体硬件平台。

多媒体计算机的主要硬件除了常规的硬件如主机、软盘驱动器、硬盘驱动器、显示器和打印机之外,还要有音频信息处理硬件、视频信息处理硬件及光盘驱动器等。

(1) 音频卡(sound card)又称声卡,用于处理音频信息。它可以把话筒、录音机和电子乐器等输入的声音信息进行模数转换(A/D)、压缩等,也可以把经过计算机处理的数字化的声音信号通过还原(解压缩)、数模转换(D/A)后用音箱播放出来,或者用录音设备记录下来。

目前,市场上声卡的种类很多,如较流行的 Creative Labs 的声霸卡(sound blaster)系列。声卡的分类主要根据其数据采样量位数来确定,通常分为 8 位、16 位和 32 位等。位数越多,其量化精度越高,音质就越好。音频卡的关键技术包括数字音频、音乐合成(FM 合成和波形表合成)、MIDI(数字音乐的国际标准,指乐器数字接口)和音效。数字音频部分具有 44.1kHz 的采样率,8 位以上的分辨率。具有录音和播放声音信号的功能,同时具有压缩采样信号的能力。最常用的压缩方法是自适应脉冲编码调制。数字音

频的实现有不同的方法和芯片,大多数采用的是 CODEC 芯片,它具有硬件压缩功能。部分采用的是 DSP+ADC(数字信号处理芯片+A/D 转换器)方法,它利用软件的方法压缩数字音频信号。波形表合成使用 DSP 技术,它要求大容量的 ROM,以获得高质量的演奏效果。

(2) 视频卡(video card)也称为显示卡,是基于 PC 机的一种多媒体视频信号处理平台,用来支持视频信号的输入与输出。它可以采集视频源、音频源和激光视盘机、录像机以及摄像机等设备的信息,经过编辑或特技等处理而产生非常精美的画面。还可以对这些画面进行捕捉、数字化、冻结、存储、压缩和输出等操作。画面的修整、像素显示调整以及缩放功能等都是视频卡支持的标准功能。视频卡可细分为视频捕捉卡、视频处理卡、视频播放卡以及 TV 编码器等专用卡,其功能是连接摄像机、VCR 影碟机和 TV 等设备,以便获取、处理和表现各种动画和数字化视频媒体。

(3) 光盘驱动器分为只读光盘驱动器(CD-ROM、DVD-ROM)和可读写光盘驱动器(CD-R,CD-RW)。可读写光盘驱动器又称刻录机。CD-ROM 及 DVD-ROM 光盘驱动器是目前多媒体应用的主要设备,主要用于存储大量的影像、声音、动画、程序数据和高分辨率图像信息。CD-ROM 可为多媒体计算机提供 650MB 的存储容量,DVD-ROM 的存储量更大,双面可达 17GB。CD-ROM 驱动器价格便宜,对广大用户来说已经是必须配置的设备。CD-ROM 的速度和兼容性也是重要的技术指标。单倍速(150KB/s 驱动器)是播放 CD-Audio 的最低要求,而 10 倍速的驱动器数据传输率可达 1500KB/s。

(4) 扫描仪用于把摄影作品、绘画作品或其他印刷材料上的文字、图像甚至实物扫描输入到计算机中,进而对这些图像信息进行加工处理、管理、使用、存储和输出。扫描仪是获取图像的一种较简单的方法,现在已成为较流行的图像输入设备。扫描仪的种类很多,常用的有手持式扫描仪、滚筒式扫描仪和平板式扫描仪等。

- 手持式扫描仪通过手工移动扫描仪扫描图稿。一般来说,扫描宽度为 10.5,分辨率为 300~2400dpi,色彩深度为 18~24bit。其特点是体积小、重量轻和价格低,但扫描的图像容易失真。
- 滚筒式扫描仪通过软件控制扫描仪自动完成扫描过程。其扫描宽度大,可以用作大幅面图稿的输入,分辨率为 600~4800dpi,色彩深度为 24~36bit。其特点是速度快、精度高、操作简单以及连续扫描(自动进纸)。
- 平板式扫描仪通过软件控制扫描仪自动完成扫描过程,扫描宽度为 A4 幅面,分辨率为 600~9600dpi,色彩深度为 24~36bit。其特点是速度快、精度高和操作简单,但其价格比较高。

(5) 光学字符阅读器是一种文字自动输入设备。它通过扫描或摄像等光学输入方法从纸介质上获取文字的图像信息,利用各种模式识别算法分析文字的形态特征,判断文字

的标准编码,并按通用格式存储在文本文件中。

(6) 触摸屏是一种随多媒体技术发展而使用的输入设备。当手指点在屏幕上的菜单或图符等图形/图像按钮时,产生触摸信号。该信号经过变换后成为计算机可以处理的命令,从而实现人机交互。

(7) 数字化仪是一种图形输入设备,它由平板加上连接的手动定位装置组成,主要用于输入地图和气象图等线型图。可以通过手动定位笔方便地获得每个线段的起始坐标,从而实现线型图输入。

(8) 操纵杆是一种提供位置信息的输入设备,可支持其他定点输入设备,将它们(数字化仪和光笔等)输入的位置和按键信息提供给应用程序。在多媒体计算机上,操纵杆常用作游戏控制器,用来操纵电子游戏。它可以以模拟信号方式工作,也可以以数字信号方式工作,并可兼容各种游戏模式。

(9) 绘图仪是一种图形输出设备。投影仪是一种将计算机输出的视频信号投影到幕布上的设备。

6.6.2 多媒体软件系统

多媒体计算机软件系统主要包括多媒体操作系统、多媒体应用软件的开发工具和多媒体应用软件。

1. 多媒体操作系统

多媒体操作系统必须具备对多媒体环境下的各个任务进行调度和管理,支持多媒体应用软件运行,对多媒体声像及其他多媒体信息进行控制和实时处理,支持多媒体的输入输出及相应的软件接口,对多媒体数据和多媒体设备的管理和控制以及图形用户界面管理等功能。也就是说,它能够像一般操作系统处理文字、图形和文件那样去处理音频、图像和视频等多媒体信息。并能够对 CD-ROM 驱动器、录像机、MIDI 设备、数码相机和扫描仪等各种多媒体设备进行控制和管理。传统计算机所使用操作系统或多或少均支持多媒体,例如,Windows 3.1 在支持多媒体方面要比 DOS 强,但它们还不是真正意义上的多媒体操作系统。随着多媒体技术的发展,使得传统的操作系统增加了许多适应多媒体的内容,并研制出具备多媒体功能的操作系统。Apple 公司的 Quick Time 以及当前流行的 Windows 系列产品 Windows 98、Windows ME、Windows 2000 和 Windows XP 等都具备多媒体功能。

2. 多媒体创作工具软件

多媒体创作工具是在多媒体操作系统之上的系统软件,它提供了建立多媒体节目的构件和框架的功能,可以实现媒体的组接和交互跳转功能。通常,多媒体创作工具软件是用于开发多媒体应用程序的应用工具软件,帮助应用开发人员提高开发工作效率。它们

大体上都是一些应用程序生成器,将各种媒体素材按照超文本节点和链结构的形式进行组织,形成多媒体应用系统。

多媒体创作工具软件按照组织方式与数据管理方式大致上可分为以下几类。

1) 页面模式的创作工具

这类创作工具按照类似于书的页面的方式来组织和管理,具有出色的超文本和超媒体功能。这类工具主要有以下两种。

(1) Power Point: 它是一种最简单实用的基于页面的创作工具软件。每一个画面可以看作一个页面,可以分别进行生成、编辑和排列。

(2) Tool Book: 脚本模式的应用程序可以被想象成一本有许多页的书。每一页是展示在它自己窗口中的一个画面,其中包含许多多媒体对象(图形及按钮等)和大量的交互信息。这里所谓的页是比 Power Point 更丰富的一种结构,可以在一页之内进行交互。

2) 时序模式的创作工具

这类创作工具按照时间顺序来组织数据或事件。这种顺序的排列一般是以帧为单位的。如同电影编导过程,可以精确控制什么时间播放什么镜头。这种工具适合于处理动画、视频图像等。这类创作工具主要有以下两种。

(1) Director: 它是以总谱为基础,以角色和帧为对象的多媒体创作工具。总谱可以看作电影中导演脚本的形象表现。角色是指所有需要单独控制的素材,包括声音、文本、图形、图像、调色板、视频、动画和按钮等,均作为角色统一管理。每个角色都有自己的属性,可以通过多种方法进行控制。

(2) Flash: 它是广泛应用于网页交互多媒体动画设计的工具软件,具有提供各种创建原始动画素材的方式。可将图形图像生成逐帧动画,支持多种文件格式(能导入导出位图、视频以及音频等媒体文件)和通用的浏览器,具有强大的交互功能。

3) 图标模式的创作工具

这类创作工具以对象或事件的顺序来组织数据。以流程线为主干,将各种媒体逐个组接在流线中。通常,一个应用程序结构的建立是通过从图标板中拖曳图标放入应用程序工作区间进行的,并把它们联系起来产生一个应用程序的逻辑结构。这是一种特别适合于一般用户使用的创作方式。Authorware 就是这种工具。

Authorware 是用来创作与发行交互式学习的软件开发工具,众多的开发者用它来进行教育训练、教学和多媒体应用软件的开发。其应用程序结构的建立是通过从图标板中拖曳图标放入应用程序工作区间的,并把它们联系起来产生一个应用程序的逻辑结构。这是一种特别适合于一般用户使用的创作方式。它支持 ActiveX、Oracle Video Server、Flash、媒体元素浏览器以及许多图形/图像格式(BMP、DIB、GIF、JPEG、Photoshop 3、PNG 和 TARGA 等),并能够以这些图形/图像的原始格式来进行处理。

4) 窗口模式的创作工具

一个窗口是屏幕上一个与用户交互的对象。窗口中的所有控件和对象都通过窗口接受控制。Visual Basic、Visual C++ 和 Delphi 等编程语言都是基于程序语言的集成包,提供窗口及其对象的图形创作方式。

3. 多媒体素材编辑软件

在多媒体应用中,很重要的一个环节是制作所需要的各种媒体素材。多媒体素材编辑软件用于采集、整理和编辑各种媒体数据。多媒体编辑软件主要有两种。

1) 文本工具

其功能主要是文字处理(编辑、排版和识别等),常用的字处理工具有 WPS、Notebook(记事本)、Writer(写字板)、Word 和 OCR(光学字符识别)等。

2) 图形/图像工具

主要功能包括图形/图像显示、图形/图像编辑、图像压缩、图像捕捉以及图形/图像素材库,常用的图形/图像处理工具列举如下。

(1) Photoshop: 用于图像设计、编辑与处理,其功能强大,是使用最多的一种图形/图像工具软件。

(2) Illustrator: 主要用于产品包装、网页图形、演示、标志设计、字形处理以及工程绘图等。

(3) PhotoDeluxe(中文版): 主要用于数码相片处理。

(4) PageMaker(中文版): 它是一个专业排版与图形制作的工具软件。

(5) CorelDraw: 它是一种矢量图形/图像软件,广泛用于企业形象设计、广告设计和印刷设计等。

(6) AutoCAD(中文版): 它是一种矢量图形/图像软件,广泛用于机械设计和建筑设计等。

(7) Freehand: 它是一种矢量图形制作软件。

(8) 3DS Max: 它是一种功能强大,广泛使用三维图形图像编辑软件。

(9) Screen Thief: 它是一种支持静态抓图的工具,并支持 BMP、GIF、PCX 和 RLE 等多种文件格式。

3) 动画工具

主要包括动画显示、动画编辑以及动画素材库等功能,常用的动画工具软件如下所述。

(1) GIF Construction Set: 它是一种能够处理和创建 GIF 格式文件的功能强大的工具,能快速、专业地为网页创建透明、交错和活动的 GIF 文件。

(2) Xara3D: 它是一种 3D 图形工具软件,可用于制作高质量的三维动画,全面支持

中文。

4) 视频工具

主要包括视频显示、视频编辑、视频压缩、视频捕捉以及视频素材库等功能,常用的动画工具软件如下所述。

(1) Media Studio Pro(中文版): 它是一个功能强大的专业桌面数码视频编辑软件,提供一套视频捕捉、编辑、转换及特效制作等艺术解决方案;

(2) Premiere: 它是一种功能强大的视频编辑软件,提供编辑、特技处理和剪辑等视频编辑功能,以及编辑静态图像和声音的工具。

5) 音频工具

主要包括音频播放、音频编辑、音频录制以及声音素材库等 4 个功能,常用的音频工具软件列举如下。

(1) CoolEditPro: 它是一种功能很强的数字音频处理软件,提供多轨编辑和数字信号处理(DSP)等功能,支持 WAV、MP3、AU、MPEG、MOV 以及 AVI 等众多的音频格式。

(2) GoldWave: 它是一种小巧好用的数码录音及编辑软件,除具有许多效果处理外,还有文件格式转换功能,支持多种声音格式,如 WAV、MP3、AU、MPEG、MOV 及 AVI 等。

(3) Cake Walk Pro Audio: 它是目前流行的专业音乐制作工具软件,可以用来作曲、配器、演奏、录音及合成等,功能十分强大。

6) 播放工具

主要用于显示、浏览或播放图像、音频及视频等多媒体数据。常用的播放工具软件列举如下。

(1) Media Player(媒体播放器): Windows 操作系统内置的媒体播放器,主要用于控制多媒体设备并播放多媒体文件,如声音、动画及视频等;

(2) ACDSee: 它是一种图像浏览工具,支持 BMP、GIF、JPG 及 TGA 等各种常见的图形/图像文件格式,图片打开速度极快。

4. 多媒体应用软件

多媒体应用软件是具体实用的应用程序及演示软件,它是直接面向用户或信息发送与接收的软件。这类应用软件用户界面十分友好,用户只要根据应用软件给出的操作命令,通过最简单的操作便可使用这些软件(根据多媒体系统终端用户要求而开发的应用软件)。例如,特定的专业信息管理系统,语音/Fax/数据传输调制管理应用系统,多媒体监控系统,CD-ROM 光盘播放软件,各种多媒体 CAI 软件,以及各种游戏软件等。在人们探讨应用多媒体技术解决自己面临的应用实际问题时,设计建造出各式各样的应用软件

系统,使最终用户运用多媒体系统时能够方便、易学 and 好用。除面向终端用户而制定的应用软件外,另一类是面向某一个领域的用户应用软件系统。这实际上是面向大规模用户的系统产品,如多媒体会议系统、电视点播服务(VOD)以及医用、家用、军用和工业应用系统等。

6.7 虚拟现实的概念

虚拟现实(virtual reality, VR)将是继多媒体、计算机网络之后,最具有应用前景的一种技术。它可应用于建模与仿真、科学计算可视化、设计与规划、教育与训练、医学、艺术与娱乐等方面。虚拟现实技术是一项综合的技术,涉及计算机科学、电子学、心理学、计算机图形学、人机接口技术、传感技术及人工智能技术等。这种技术的特点在于,运用计算机对现实世界进行全面仿真,创建与现实社会类似的环境,通过多种传感设备使用户“投入”到该环境中,实现用户与该环境的直接自然交互。

1. 重要特征

(1) 多感知:就是说除了一般计算机所具有的视觉感知外,还有听觉感知、力觉感知、触觉感知、运动感知,甚至包括味觉感知和嗅觉感知等。理想的虚拟现实就是应该具有人所具有的感知功能。

(2) 沉浸(又称临场感):是指用户感到作为主角存在于模拟环境中的真实程度。理想的模拟环境应该达到使用户难以分辨真假的程度。

(3) 交互:是指用户对模拟环境内物体的可操作程度和从环境得到反馈的自然程度(包括实时性)。例如,用户可以用手去直接抓取环境中的物体,这时手应有握着东西的感觉,并可以感觉物体的重量,视场中的物体也会随着手的移动而移动。

2. 生成技术

将现实世界的多维信息映射到计算机的数字空间,并生成相应的虚拟世界,主要包括基本模型构建、空间跟踪、声音定位、视觉跟踪和视点感应等关键技术。

(1) 基本模型构建:基本模型的构建是应用计算机技术生成虚拟世界的基础,它将真实世界的对象物体在相应的三维虚拟世界中重构,并根据系统需求保存部分物理属性。模型构建首先要建立对象物体的几何模型,确定其空间位置和几何元素的属性。例如,通过 CAD/CAM 或二维图纸构建产品或建筑的三维几何模型。通过卫星、遥感或航拍照片构造大型虚拟战场。为了增强虚拟环境的真实感、物理特性和行为规则,建模须表现出对象物体的质量、动量和材料等物理特性,并在虚拟环境中遵循一定的运动和动力学规律。

(2) 空间跟踪:虚拟环境的空间跟踪主要是通过头盔显示器、数据手套和数据衣等交互设备上的空间传感器,确定用户的头、手、躯体或其他操作物在三维虚拟环境中的位

置和方向。跟踪系统一般由发射器、接收器和电子部件组成。目前的跟踪系统有电磁、机械、光学和超声等几类。例如,数据手套是 VR 系统常用的一种人机交互设备,通过手指上的弯曲传感器、扭曲传感器和手掌上的弯度传感器、弧度传感器来确定手及关节的位置和方向,从而实现环境中的虚拟手及其对虚拟物体的操纵。

(3) 声音跟踪:利用不同声源的声音到达某一特定地点的时间差、相位差和声压差等进行虚拟环境的声音跟踪。声音跟踪一般包括若干个发射器、接受器和控制单元。它可以与头盔显示器相连,也可以与数据衣、数据手套等其他设备相连。

(4) 视觉跟踪与视点感应:视觉跟踪技术使用从视频摄像机到 X-Y 平面阵列,根据周围光或者跟踪光在图像投影平面不同时刻和不同位置上的投影,计算被跟踪对象的位置和方向。视觉跟踪的实现必须考虑精度和操作范围间的折中选择,采用多发射器和多传感器的设计能增强视觉跟踪的准确性,但使系统变得复杂并且成本高。视点感应需要与显示技术相结合,采用多种定位方法(眼罩定位、头盔显示、遥视技术和基于眼肌的感应技术)可确定用户在某一时刻的视线。

3. 反馈方式

在虚拟环境中获取视觉、听觉、力觉和触觉感官认知等关键技术,是保证虚拟世界中的事物所产生的各种刺激以尽可能自然的方式反馈给用户。

(1) 视觉感知:虚拟环境中,大部分具有一定形状的物体或现象可以通过多种途径使用户产生真实感很强的视觉感知。CRT 显示器、大屏幕投影、多方位电子墙、立体眼镜和头盔显示器(HMD)等是 VR 系统中常见的显示设备。不同的头盔显示器采用不同的显示技术,根据光学图像的提供方式,头盔显示设备可分为投影式和直视式。

(2) 听觉感知:听觉是仅次于视觉的感知途径。虚拟环境的声音效果,可以弥补视觉效果的不足,增强环境逼真度。用户所感受的三维立体声音,有助于用户在操作中对声音定位。

(3) 力觉和触觉感知:能否让参与者产生“沉浸”感的关键因素之一是参与者能否在操纵虚拟物体的同时,感受到虚拟物体的反作用力,从而产生触觉和力觉感知。例如,当用手扳动虚拟驾驶系统的汽车档位杆时,手能感觉到档位杆的震动和松紧。力觉感知主要由计算机通过力反馈手套、力反馈操纵杆对手指产生运动阻尼,从而使用户感受到作用力的方向和大小。由于人的力觉感知非常敏感,对力反馈装置的精度要求很高。如果没有触觉反馈,当用户接触到虚拟世界的某一物体时容易使手穿过物体。解决这种问题的有效方法是在用户的交互设备中增加触觉反馈。触觉反馈主要是基于视觉、气压感、震动触感、电子触感和神经肌肉模拟等方法来实现的。

4. 分类

普通意义上的虚拟现实需要大型计算机、头盔式显示器、立体眼镜、数据手套、洞穴式

投影以及密封仓等一系列传感辅助设施来实现的一种三维现实。人们可以通过这些设施以自然的方式(如头的转动、手的运动等)向计算机送入各种动作信息,并且通过视觉、听觉以及触觉设施使人们得到三维的视觉、听觉及触觉等感觉世界。随着人们的动作不同,这些感觉也随之改变。事实上,随着科学技术的飞速发展,虚拟现实技术出现了多样化的发展趋势,虚拟现实技术不仅仅是指那些戴着头盔和手套等技术,而且还应该包括一切与之有关的具有自然模拟、逼真体验的技术与方法,它的根本目标就是达到真实体验和基于自然技能的人机交互,而能够达到或者部分达到这样目标的系统就称为虚拟现实系统。根据用户参与 VR 的不同形式以及沉浸的不同程度,可以把各种类型的虚拟现实技术大致划分为 4 类:

1) 桌面虚拟现实

桌面虚拟现实利用个人计算机和低级工作站进行仿真,将计算机的屏幕作为用户观察虚拟境界的一个窗口。通过各种输入设备实现与虚拟现实世界的充分交互,这些外部设备包括鼠标、追踪球和力矩球等。它要求参与者使用输入设备,通过计算机屏幕观察 360°范围内的虚拟境界,并操纵其中的物体。但在这种情况下,参与者缺少完全的沉浸,因为他们仍然会受到周围现实环境的干扰。桌面虚拟现实的最大特点是缺乏真实的现实体验,但成本相对较低,因而应用比较广泛。常见的桌面虚拟现实技术有基于静态图像的虚拟现实 Quick Time VR、虚拟现实造型语言 VRML、桌面三维虚拟现实以及 MUD 等。

2) 完全沉浸的虚拟现实

高级虚拟现实系统提供完全沉浸的体验,使用户有一种置身于虚拟境界中的感觉。它利用头盔式显示器或其他传感设备,把参与者的视觉、听觉和其他感觉封闭起来,提供一个新的、虚拟的感觉空间,并利用位置跟踪器、数据手套、其他手控输入设备和声音等使得参与者产生一种身临其境、全心投入和沉浸其中的感觉。常见的沉浸式系统有基于头盔式显示器的系统、投影式虚拟现实系统以及远程存在系统等。

3) 增强现实性的虚拟现实

增强现实性的虚拟现实不仅利用虚拟现实技术来模拟现实世界、仿真现实世界,而且还要利用它来增强参与者对真实环境的感受。

4) 分布式虚拟现实

分布式虚拟现实系统是基于网络的虚拟环境。在这个环境中,位于不同物理环境位置的多个用户或多个虚拟环境通过网络相连接,或者多个用户同时参加一个虚拟现实环境,通过计算机与其他用户进行交互,并共享信息。在分布式虚拟现实系统中,多个用户可通过网络对同一虚拟世界进行观察和操作,以达到协同工作的目的。

第7章 数据库技术基础

数据库系统本质上是一个用计算机存储记录的系统。数据库管理系统是位于用户与操作系统之间的一层数据管理软件,其基本目标是提供一个可以方便地、有效地存取数据库信息的环境。数据库就是信息的集合,它是收集计算机数据的仓库或容器,系统用户可以对这些数据执行一系列操作。设计数据库系统的目的是为了管理大量信息,给用户提提供数据的抽象视图,即系统隐藏了有关数据存储和维护的某些细节。对数据的管理涉及到信息存储结构的定义,信息操作机制,安全性保证,以及多用户对数据的共享问题。

本章主要介绍一些背景知识和基本概念,使读者了解数据库的基本内容,形成数据库系统的总体框架,了解数据库系统在计算机系统中的地位,以及数据库系统的功能。

7.1 基本概念

7.1.1 数据库与数据库管理系统

数据是描述事物的符号记录,它具有多种表现形式,可以是文字、图形、图像、声音和语言等。信息是现实世界事物的存在方式或状态的反映。信息具有可感知、可存储、可加工、可传递和可再生等自然属性。信息已是社会各行各业不可缺少的资源,这也是信息的社会属性。数据是经过组织的比特的集合,而信息是具有特定释义和意义的数据。

数据库系统(database system,DBS)广义上讲是由数据库、硬件、软件和人员组成的,其中管理的对象是数据。数据是经过组织的比特的集合,而信息是具有特定释义和意义的数据。

(1) 数据库(database,DB)是指长期储存在计算机内的,有组织的,可共享的数据的集合。数据库中的数据按一定的数学模型组织、描述和储存,具有较小的冗余度,较高的数据独立性和易扩展性,并可为各种用户共享。

(2) 硬件:构成计算机系统的各种物理设备,包括存储数据所需的外部设备。硬件的配置应满足整个数据库系统的需要。

(3) 软件:包括操作系统、数据库管理系统及应用程序。数据库管理系统(database management system,DBMS)是数据库系统的核心软件,是在操作系统的支持下工作,解决如何科学地组织和储存数据,如何高效地获取和维护数据的系统软件。其主要功能包

括：数据定义功能、数据操纵功能、数据库的运行管理和数据库的建立与维护。

(4) 人员：主要有 4 类。第一类为系统分析员和数据库设计人员：系统分析员负责应用系统的需求分析和规范说明，他们和用户及数据库管理员一起确定系统的硬件配置，并参与数据库系统的概要设计。数据库设计人员负责数据库中数据的确定、数据库各级模式的设计。第二类为应用程序员，负责编写使用数据库的应用程序。这些应用程序可对数据进行检索、建立、删除或修改。第三类为最终用户，他们利用系统的接口或查询语言访问数据库。第四类用户是数据库管理员 (data base administrator, DBA)，负责数据库的总体信息控制。DBA 的具体职责包括：决定数据库中的信息内容和结构，决定数据库的存储结构和存取策略，定义数据库的安全性要求和完整性约束条件，监控数据库的使用和运行，负责数据库的性能改进、数据库的重组和重构，以提高系统的性能。

7.1.2 数据库技术的发展

计算机的主要应用之一就是数据处理，将大量的信息以数据的形式存放在磁盘上。数据处理是指对各种数据进行收集、存储、加工和传播的一系列活动。数据管理是数据处理的中心问题，是对数据进行分类、组织、编码、存储、检索和维护。数据管理技术的发展经历了 3 个阶段：人工管理、文件系统和数据库系统阶段。

1. 人工管理阶段

早期的数据处理都是通过手工进行的，因为当时的计算机主要用于科学计算。计算机上没有专门管理数据的软件，也没有诸如磁盘之类的设备来存储数据，那时应用程序和数据之间的关系如图 7-1 所示。



图 7-1 应用程序和数据的关系

这种数据处理具有以下几个特点：

(1) 数据量较少。数据和程序一一对应，即一组数据对应一个程序，数据面向应用，独立性很差。由于应用程序所处理的数据之间可能会有一定的关系，故程序和程序之间就会有大量的重复数据。

(2) 数据不保存。因为该阶段计算机主要用于科学计算，一般不需要将数据长期保存，只在计算一个题目时，才将数据输入计算机，算完题后得到计算结果即可。

(3) 没有软件系统对数据进行管理。程序员不仅要规定数据的逻辑结构，而且还要在程序中设计物理结构，包括存储结构的存取方法、输入输出方式等。也就是说，数据对程序不具有独立性，一旦数据在存储器上改变物理地址，就需要改变相应的用户程序。

手工处理数据有两个特点：第一，应用程序和数据之间的依赖性太强，不独立；第二，数据组和数据组之间可能有许多重复数据，造成数据冗余。

2. 文件系统阶段

20 世纪 50 年代中期以后,计算机的硬件和软件得到飞速发展,计算机不再只用于科学计算的单一任务,而是可以做一些非数值数据的处理。又由于大容量的磁盘等辅助存储设备的出现,使得专门管理辅助存储设备上的数据的文件系统应运而生,它是操作系统中的一个子系统。在文件系统中,按一定的规则将数据组织成为一个文件,应用程序通过文件系统对文件中的数据进行存取和加工。文件系统对数据的管理,实际上是通过应用程序和数据之间的一种接口实现的,如图 7-2 所示。

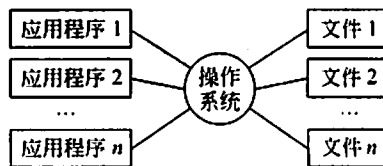


图 7-2 应用程序与文件的关系

文件系统的最大特点是解决了应用程序和数据之间的一个公共接口问题,使得应用程序采用统一的存取方法来操作数据。在文件系统阶段中数据管理的特点如下:

- 数据可以长期保留,数据的逻辑结构和物理结构有了区别,可以按名访问,不必关心数据的物理位置,由文件系统提供存取方法。
- 数据不属于某个特定的应用,即应用程序和数据之间不再是直接的对应关系,可以重复使用。但文件系统只是简单地存取数据,相互之间并没有有机的联系,即数据存取依赖于应用程序的使用方法,不同的应用程序仍然很难共享同一数据文件。
- 文件组织形式的多样化,有索引文件、链接文件和散列文件等。但文件之间没有联系,相互独立,数据间的联系要通过程序去构造。

文件系统具有如下缺点:

- 数据冗余度大:文件与应用程序密切相关。相同的数据集合在不同的应用程序中使用时,经常需要重复定义、重复存储。例如工厂中人事处管理的职工人事档案,生产科考勤系统管理的职工出勤情况,所用到的数据很多都是重复的。相同的数据不能被共享,必然导致数据的冗余。
- 数据不一致性:由于相同数据的重复存储,单独管理,所以给数据的修改和维护带来难度,容易造成数据的不一致。例如,人事处修改了某个职工的信息,但生产科该职工的相应信息没有修改,造成同一个职工的信息在不同的部门结果不一样。
- 数据联系弱:文件系统中数据组织成记录,记录由字段组成,记录内部存在一定的结构。但文件之间是孤立的,从整体上看没有反映现实世界事物之间的内在联系,因此很难对数据进行合理的组织以适应不同应用的需要。

3. 数据库系统阶段

数据库系统是由计算机软件 and 硬件资源组成的系统,它实现了有组织地、动态地存储大量的关联数据,便于多用户访问。它与文件系统的重要区别是数据的充分共享、交叉访问以及与应用程序的高度独立性。

数据库系统阶段数据管理的特点如下:

(1) 采用复杂的数据模型表示数据结构。数据模型不仅描述数据本身的特点,还描述数据之间的联系。数据不再面向某个应用,而是面向整个应用系统。数据冗余明显减少,实现了数据共享。

(2) 有较高的数据独立性。数据库也是以文件方式存储数据的,但它是一种更高级的数据组织形式,由 DBMS 负责应用程序和数据库之间的数据存取。DBMS 对数据的处理方式和文件系统不同,它把所有应用程序中使用的数据以及数据间的联系汇集在一起,以便于应用程序查询和使用。这一阶段程序和数据的关系如图 7-3 所示。

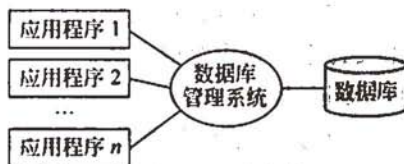


图 7-3 应用程序与数据库的关系

数据库系统与文件系统的区别是:数据库对数据的存储是按照同一结构进行的,不同的应用程序都可以直接操作这些数据(即对于应用程序的高度独立性)。数据库系统对数据的完整性、惟一性和安全性都提供一套有效的管理手段(即数据的充分共享性)。数据库系统还提供管理和控制数据的各种简单操作命令,使用户编写程序时容易掌握(即操作方便性)。

7.2 数据模型

7.2.1 数据模型的基本概念

模型是对现实世界特征的模拟和抽象,数据模型是对现实世界数据特征的抽象。对于具体的模型人们并不陌生,如航模飞机、地图和建筑设计沙盘都是具体的模型。从事物的客观特性到计算机里的具体表示经历了现实世界、信息世界和机器世界 3 个数据领域。

(1) 现实世界:现实世界的的数据就是客观存在的各种报表、图表和查询格式等原始数据。计算机只能处理数据,所以首先要解决的问题是按用户的观点对数据和信息建模,

即抽取数据库技术所研究的数据,分门别类,综合出系统所需要的数据。

(2) 信息世界:是现实世界在人们头脑中的反映,人们用符号、文字记录下来。在信息世界中,数据库常用的术语是实体、实体集、属性和码。

(3) 机器世界:是按计算机系统的观点对数据建模,即现实世界的问题如何表达为信息世界的问题,而信息世界的问题又如何在具体的机器世界中表达。机器世界中描述数据的术语有字段、记录、文件和记录码。

信息世界与机器世界相关术语的对应关系如下所示。

- 属性与字段:属性描述的是实体某方面的特性,字段是标记实体属性的命名单位。例如,用“书号、书名、作者名、出版社和日期”5个属性描述书的特性,对应应有5个字段。
- 实体与记录:实体表示客观存在并能区别的事物(如一个学生、一本书);记录是字段的有序集合,一般一条记录描述一个实体。例如“10001, DATABASE SYSTEM CONCEPTS, China Machine Press, 2000-2”描述的是一个实体,对应一条记录。
- 码与记录码:码是能惟一区分实体的属性或属性集,记录码是惟一标识文件中每条记录的字段或字段集。
- 实体集与文件:实体集是具有共同特性的实体的集合,文件是同一类记录的汇集。例如所有学生构成了学生实体集,而所有学生记录组成了学生文件。
- 实体型与记录型:实体型是属性的集合,如表示学生学习情况的属性的集合为实体型(Sno, Sname, Sage, Grade, SD, Cno……);记录型是记录的结构定义。

7.2.2 数据模型的三要素

数据库结构的基础是数据模型,用来描述数据的一组概念和定义。数据模型的三要素是数据结构、数据操作以及数据的约束条件。

例如,以大家熟悉的文件系统为例,它所包含的概念有文件、记录和字段。

- 数据结构和约束条件:对每个字段定义数据类型和长度。
- 数据操作:打开、关闭、读和写等文件操作。

上述是一个简单的数据模型,没有描述数据间的联系。

(1) 数据结构 是所研究的对象类型的集合,是对系统静态特性的描述。

(2) 数据操作 对数据库中各种对象(型)的实例(值)允许执行的操作及操作规则的集合。这些操作可以是检索、插入、删除和修改,操作规则有优先级别等。数据操作是对系统动态特性的描述。

(3) 数据的约束条件 是一组完整性规则的集合。也就是说,具体的应用数据必须

遵循特定的语义约束条件,以保证数据的正确、有效和相容。例如,某单位人事管理中,要求在职“男”职工的年龄必须大于 18 岁小于 60 岁,工程师的基本工资不能低于 1500 元,每个职工可担任一个工种,这些要求可以通过建立数据的约束条件来实现。

最常用的数据模型分为概念数据模型和基本数据模型。

- 概念数据模型:也称信息模型,是按用户的观点对数据和信息进行建模,是现实世界到信息世界的第一层抽象,强调其语义表达功能,易于用户理解,是用户和数据库设计人员交流的语言,主要用于数据库设计。在这类模型中,最著名的是实体联系模型,简称 E-R 模型。
- 基本数据模型:它是按计算机系统的观点对数据进行建模,是现实世界数据特征的抽象,用于实现 DBMS。基本的数据模型有层次模型、网状模型和关系模型。

值得注意的是,目前出现了许多数据库应用的新领域。采用基本数据模型有一定的局限性,所以面向对象模型(object oriented model)越来越受到关注。

7.2.3 E-R 模型

实体联系模型简称 E-R 模型,所采用的 3 个主要概念是:实体、联系和属性。E-R 模型是软件工程设计中的一个重要方法,因为它接近于人的思维方式,容易理解并且与计算机无关,所以用户容易接受。但是,E-R 模型只能说明实体间的语义联系,还不能进一步地说明详细的数据结构。遇到实际问题,通常应先设计一个 E-R 模型,然后再把它转换成计算机能接受的数据模型。

1. 实体

实体是现实世界中可以区别于其他对象的“事件”或“物体”。例如,企业中的每个人都是一个实体。每个实体由一组特性(属性)来表示,其中的某一部分属性可以惟一地标识实体,如职工号。实体集是具有相同属性的实体集合。例如,学校所有教师具有相同的属性,因此教师的集合可以定义为一个实体集。学生也具有相同的属性,因此学生的集合可以定义为另一个实体集。

2. 联系

实体的联系分为实体内部的联系和实体与实体之间的联系。实体内部的联系反映数据在同一记录内部各字段间的联系,这里着重讨论实体集之间的联系。

1) 两个不同实体集之间的联系

两个不同实体集之间存在一对一、一对多和多对多的联系。

- 一对一 指实体集 E_1 中的一个实体最多只与实体集 E_2 中的一个实体相联系。记为 1:1。
- 一对多:表示实体集 E_1 中的一个实体可与实体集 E_2 中的多个实体相联系。记

为 $1:n$ 。

- 多对多：表示实体集 E_1 中的多个实体可与实体集 E_2 中的多个实体相联系。记为 $m:n$ 。

图 7-4 表示两个不同实体集之间的联系。其中：

(1) 电影院里一个座位只能坐一个观众，因此观众与座位之间是一个 $1:1$ 的联系。设联系名为“V_S”，用 E-R 图表示如图 7-4(a) 所示。

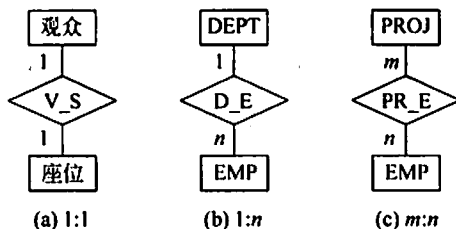


图 7-4 两个不同实体集之间的 $1:1$ 、 $1:n$ 和 $m:n$ 联系

(2) 对于部门 DEPT 和职工 EMP 实体集，若一个职工只能属于一个部门，那么，这两个实体集之间应是一个 $1:n$ 的联系。取联系名为 D_E，其 E-R 图如图 7-4(b) 所示。

(3) 对于工程项目 PROJ 和职工 EMP 实体集，若一个职工可以参加多个项目，一个项目可以让多个职工参加，那么，这两个实体集之间应是一个 $m:n$ 的联系。若联系名为“PR_E”，用 E-R 图表示如图 7-4(c) 所示。

2) 两个以上不同实体集之间的联系

两个以上不同实体集之间存在 $1:1:1$ 、 $1:1:n$ 、 $1:m:n$ 和 $r:m:n$ 的联系。图 7-5 表示 3 个不同实体集之间的联系。其中：

(1) 图 7-5(a) 表示供应商 Supp、项目 Proj 和零件 Part 之间的多对多 ($r:n:m$) 的联系，联系名为“SP_P”。表示供应商为多个项目供应多种零件，每个项目可用多个供应商供应的零件，每种零件可由不同的供应商供应的语义。

(2) 图 7-5(b) 表示病房、病人和医生之间的一对多 ($1:n:m$) 的联系，联系名为“SP_P”。表示一个特护病房有多个病人和多个医生，一个医生只负责一个病房，一个病人只属于一个病房。

注意，3 个实体集之间的多对多的联系和 3 个实体集两两之间的多对多的联系的语义是不同的。例如，供应商和项目实体集之间的“合同”联系，表示供应商签了哪几个工程合同。供应商与零件两个实体集之间的“库存”联系，表示供应商库存零件的数量。项目与零件两个实体集之间的“组成”联系，表示一个项目由哪几种零件组成。

3) 同一实体集内的二元联系

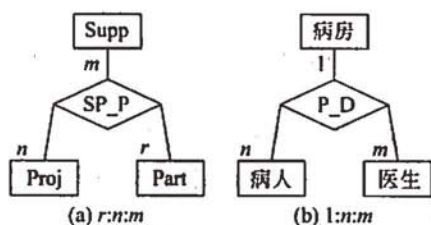


图 7-5 3 个不同实体集之间的 $r:n:m$ 和 $1:n:m$ 联系

同一实体集内的各实体之间也存在 $1:1$ 、 $1:n$ 和 $m:n$ 的联系,如图 7-6 所示。

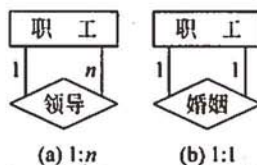


图 7-6 同一实体集之间的 $1:n$ 和 $1:1$ 联系

从图中可见,职工实体集中的领导与被领导的联系是 $1:n$ 的。但是,职工实体集中的婚姻联系是 $1:1$ 的。

3. 属性

属性是实体某方面的特性。例如,职工实体集具有职工号、姓名、年龄、参加工作时间以及通信地址等属性。每个属性都有其取值范围,如职工号为 0001~9999 的 4 位整型数,姓名为 10 位的字符串,年龄的取值范围为 18~60 等。在同一实体集中,每个实体的属性及其域是相同的,但可能取不同的值。E-R 模型中的属性有如下分类。

1) 简单属性和复合属性

简单属性是原子的、不可再分的。复合属性可以细分为更小的部分(即划分为别的属性)。用户有时希望访问整个属性,有时希望访问属性的某个成分,那么在模式设计时可采用复合属性。例如,职工实体集的通信地址可以进一步分为邮编、省、市和街道。若不特别声明,通常指的是简单属性。

2) 单值属性和多值属性

这里所举的例子中,对于一个特定的实体,定义的属性都只有单独的一个值。例如,一个特定的职工只对应一个职工号和职工姓名,这样的属性叫做单值属性。但是,在某些特定情况下,一个属性可能对应一组值。例如职工可能有 0、1、或多个亲属,那么职工的亲属的姓名可能有多,这样的属性称为多值属性。

3) NULL 属性

当实体的某个属性没有值或属性值未知时,可使用 NULL 值,表示无意义或不知道。








4) 派生属性

派生属性可以从其他属性得来。例如,职工实体集中有“参加工作时间”和“工作年限”属性,那么“工作年限”的值可以由当前时间和参加工作时间得到。这里,“工作年限”就是一个派生属性。

4. E-R 方法

概念模型中最常用的方法为实体—联系方法,简称 E-R 方法。该方法直接从现实世界中抽象出实体和实体间的联系,然后用非常直观的 E-R 图来表示数据模型。E-R 图中有如表 7-1 列出的几个主要构件。

表 7-1 E-R 图中的主要构件

构 件		说 明
矩形		表示实体集
菱形		表示联系集
椭圆		表示属性
线段		将属性与相关的实体集连接,或将实体集与联系集相连
双椭圆		表示多值属性
虚椭圆		表示派生属性
双线		表示一个实体全部参与到联系集中

本教材在介绍 E-R 图时,主码的属性以下划线标明。另外,在实体集与联系的线段上标出联系的类型。

为了叙述方便,实体集有时简称实体,联系集有时简称联系。

【例 7.1】 某学校有若干个系,每个系有若干名教师和学生。每个教师可以担任若干门课程,并参加多项项目。每个学生可以同时选修多门课程。请设计该学校教学管理的 E-R 模型,要求给出每个实体和联系的属性。

解:学校教学管理的 E-R 模型应该有 5 个实体:系、教师、学生、项目和课程。

(1) 设计各实体属性如下:

系(系号,系名,主任名)

教师(教师号,教师名,职称)

学生(学号,姓名,年龄,性别)

项目(项目号,名称,负责人)

课程(课程号,课程名,学分)

(2) 各实体之间的联系有:教师担任课程的 1:n“任课”联系,教师参加项目的 n:m

“参加”联系,学生选修课程的 $n:m$ “选修”联系,教师、学生与系之间所属关系的 $1:n:m$ “领导”联系。其中“参加”联系有一个排名属性,“选修”联系有一个成绩属性。

通过上述分析,学校教学管理的 E-R 模型如图 7-7 所示。

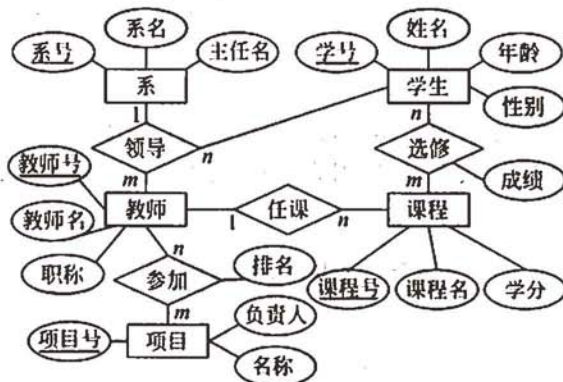


图 7-7 学校教学管理 E-R 模型

特别需要指出的是,E-R 模型强调的是语义,与现实世界的问题密切相关。这句话的意思是,尽管都是学校教学管理,但由于不同学校教学管理的方法可能会不同的语义,因此会得到不同的 E-R 模型。在数据库设计中,常用 E-R 模型来描述现实世界到信息世界的问题。因为 E-R 模型易于用户理解,是用户和数据库设计人员交流的语言。

5. 扩充的 E-R 模型

尽管用基本的 E-R 模型已足以对大多数数据库特征建模,但在某些情况下,数据库的特殊语义仅用基本 E-R 模型无法表达清楚。在这一节中,将讨论扩充的 E-R 模型,包括弱实体、特殊化、概括以及聚集等概念。

1) 弱实体

现实世界中有一种特殊的联系,这种联系代表实体间的所有(ownership)关系,例如职工与家属的联系,家属总是属于某职工的。这种实体对于另一些实体具有很强的依赖关系,即一个实体的存在必须以另一个实体为前提,这类实体被称为弱实体。

在扩展的 E-R 图中,弱实体用双线矩形框表示。图 7-8 为职工与家属的 E-R 图。



图 7-8 弱实体与依赖联系

2) 特殊化

前面已经介绍过,实体集是具有相同属性的实体集合。但在现实世界中,某些实体一方面具有一些共性,另一方面还具有各自的特殊性。这样,一个实体集按照某些特征可以区分为几个子实体。例如,学生实体集可以分为研究生、本科生和大专生等子集。我们将这种普遍到特殊的过程叫做“特殊化”。

将几个具有共同特性的实体集概括成一个更普遍的实体集的过程叫做“普遍化”。例如,可以将大专生、本科生和研究生概括为学生。还可以将学生、教师和职工概括为人。这就是从特殊到一般的过程。

设有实体集 E , 如果 S 是 E 的某些真子集的集合, 记为 $S = \{S_i | S_i \subset E, i = 1, 2, \dots, n\}$, 则称 S 是 E 的一个特殊化, E 是 S_1, S_2, \dots, S_n 的超类, S_1, S_2, \dots, S_n 称为 E 的子类。

如果 $\bigcup_{i=1}^n S_i = E$, 则称 S 是 E 的全特殊化, 否则是 E 的部分特殊化。

如果 $S_i \cap S_j = \emptyset, i \neq j$, 则 S 是不相交特殊化, 否则是重叠特殊化。

教职工实体集中的某个职工既是在职生又是教师或工人, 那么在职生、教师和工人应该是重叠特殊化。而在职生、教师和工人的集合等于教职工, 所以是全部特殊化。

在扩充的 E-R 模型中, 子类继承超类的所有的属性和联系, 但是, 子类还有自己特殊的属性和联系。例如, 研究生除了学习, 还要参加科研项目。那么, 研究生不仅要继承学生的所有属性, 还要增加学位类型、导师的属性, 并且需要增加与项目的联系。

在扩充的 E-R 图中, 超类-子类关系模型一般使用特殊化圆圈和连线的方式来表示。超类到圆圈有一条连线, 连线为双线, 则表示全特殊化, 连线为单线表示部分特殊化。双竖边矩形框表示子类。有符号“U”的线表示特殊化, 圆圈中的“d”表示不相交特殊化, 圆圈中的“O”表示重叠特殊化, 超类与圆圈用单线相连, 则表示部分特殊化。图 7-9 给出了一个特殊化应用实例。

7.2.4 层次模型

层次模型(hierarchical model)采用树型结构表示数据与数据间的联系。在层次模型中, 每一个结点表示一个记录类型(实体), 记录之间的联系用结点之间的连线表示, 并且根结点以外的其他结点有且仅有一个双亲结点。

【例 7.2】 某商场的部门、员工和商品 3 个实体的层次模型如图 7-10 所示。在该模型中, 每个部门有若干个员工, 每个部门负责销售的商品有若干种。该模型还表示部门到员工之间的联系是一对多(1 : n), 部门到商品之间也是一对多(1 : n)。

图 7-10 给出的只是 PEP 模型的“型”, 而不是“值”。在数据库中, 所谓“型”就是数据库模式, 而“值”就是数据库实例。模式是数据库的逻辑设计, 而数据库实例是给定时刻数

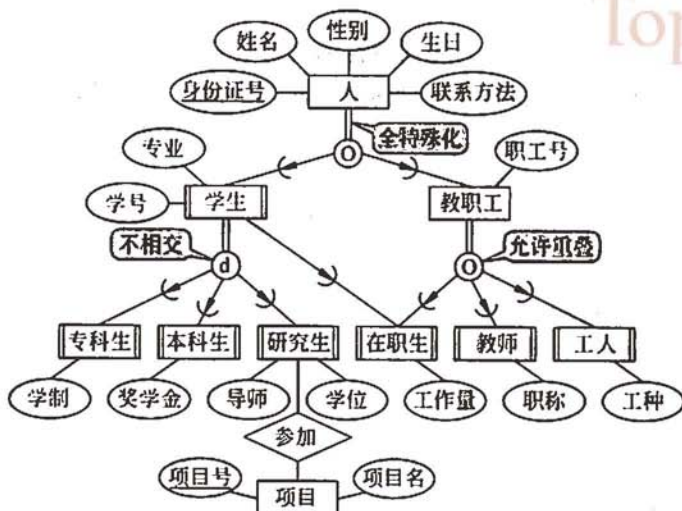


图 7-9 特殊化应用实例

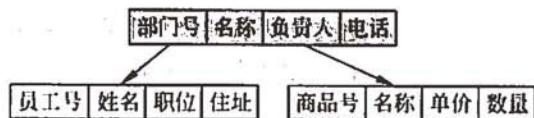


图 7-10 层次模型

数据库中数据的一个快照。图 7-11 表示销售部的一个实例。该实例表示在某一时刻销售部是由李军负责的。销售部下属有 4 个员工，负责销售的商品有 5 种。

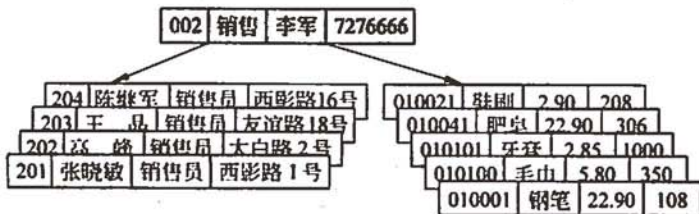


图 7-11 层次模型实例

层次模型不能直接表示多对多的联系。若要表示多对多的联系,可采用如下两种方法。

- 方法 1: 冗余结点法。两个实体多对多的联系转换为两个一对多的联系。该方法的优点是结点清晰,允许结点改变存储位置。缺点是需要额外的存储空间,有潜

在的数据不一致性。

- 方法 2: 虚拟结点分解法。将冗余结点转换为虚拟结点。虚拟结点是一个指引元,指向所代替的结点。该方法的优点是减少存储空间的浪费,避免数据不一致性。缺点是改变存储位置可能引起虚拟结点中指针的修改。

【例 7.3】 员工与商品,员工可以销售多种商品,一种商品可以由多个员工销售,所以员工与商品之间是一个多对多的联系。如图 7-12 所示。采用冗余结点法将其转换为两个一对多的联系,如图 7-13 所示。采用虚拟结点分解法,将冗余结点转换为虚拟结点,如图 7-14 所示。

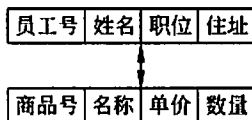


图 7-12 员工与商品的多对多联系

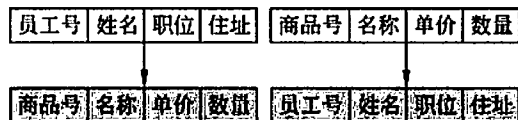


图 7-13 多对多联系转换为冗余结点表示

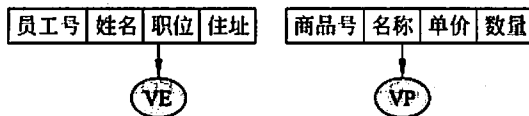


图 7-14 多对多联系转换为虚拟结点表示

- 层次模型的特点是: 记录之间的联系通过指针实现,比较简单,查询效率高。
- 层次模型的缺点是: 只能表示 $1:n$ 的联系。尽管有许多辅助手段实现 $m:n$ 的联系,但较复杂不易掌握。由于层次顺序严格和复杂,插入删除操作的限制比较多,导致应用程序编制比较复杂。1968 年美国 IBM 公司推出的 IMS 系统(信息管理系统)是典型的层次模型系统,20 世纪 70 年代在商业上得到了广泛的应用。

7.2.5 网状模型

采用网络结构表示数据与数据间的联系的数据模型称为网状模型(network model)。在网状模型中,允许一个以上的结点无双亲,一个结点可以有多个双亲。

网状模型(也称 DBTG 模型)是一个比层次模型更普遍性的数据结构,是层次模型的一个特例。网状模型可以直接地描述现实世界。因为,第一去掉了层次模型的两个限制,第二,允许两个结点之间有多种联系(称之为复合联系)。

网状模型中的每个结点表示一个记录类型(实体),每个记录类型可以包含若干个字段(实体的属性),结点间的连线表示记录类型之间一对多的联系。层次模型和网状模型的主要区别如下:

- (1) 网状模型中子女结点与双亲结点的联系不惟一,因此需要为每个联系命名。
- (2) 网状模型允许复合链,即两个结点之间有两种以上的联系,如图 7-15(a)所示。
- (3) 网状模型不能表示记录之间多对多的联系,需要引入联结记录来表示多对多的联系。

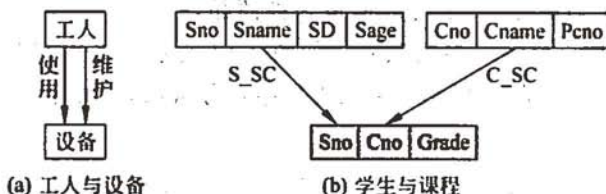


图 7-15 网状模型举例

【例 7.4】 学生、课程以及他们之间多对多的联系不能直接用网状模型表示。因为一个学生可以选若干门课,而一门课可以被多个学生选。为此,引入选课联结记录,如图 7-15(b)所示。这样,学生与选课之间的 S-SC 是一对多的联系,课程与选课之间的 C-SC 也是一对多的联系。

图 7-15 中, Sno、Sname、SD、Sage、Cno、Cname、Pcno 和 Grade 分别表示学号、姓名、系、年龄、课程号、课程名、先修课程号和成绩。

通常,网状数据模型没有层次模型那样严格的完整性约束条件,但 DBTG 在模式 DDL 中提供了定义 DBTG 数据库完整性的若干概念和语句,主要有:

- (1) 支持记录码的概念。码能惟一标识记录数据项的集合。
- (2) 保证一个联系中双亲记录和子女记录之间是一对多的联系。
- (3) 支持双亲记录和子女记录之间的某些约束条件。例如图 7-15(b)中,当插入一条选课记录“010014,100,98”时,只有当学生实体中存在学号为“010014”学生记录,课程实体存在课程号,系统才认为是合法的操作。

- 网状模型的主要优点是:能更为直接地描述现实世界,具有良好的性能,存取效率高。
- 网状模型的主要缺点是:结构复杂。例如,当应用环境不断扩大时,数据库结构就变得很复杂,不利于最终用户掌握。编制应用程序难度比较大。DBTG 模型的 DDL、DML 语言复杂,记录之间的联系是通过存取路径来实现的,因此程序员必须了解系统结构的细节,增加了编写应用程序的负担。

7.2.6 关系模型

关系模型(relation model)是目前最常用的数据模型之一。关系数据库系统采用关

系模型作为数据的组织方式,在关系模型中用表格结构表达实体集,以及实体集之间的联系,其最大特色是描述的一致性。关系模型是由若干个关系模式组成的集合。一个关系模式相当于一个记录型,对应于程序设计语言中的类型定义的概念。关系是一个实例,也是一张表,对应于程序设计语言中的变量的概念。给定变量的值随时间可能发生变化。类似地,当关系被更新时,关系实例的内容也会随时发生变化。

【例 7.5】 教学数据库的 4 个关系模式如下:

S (Sno, Sname, SD, Sage, Sex) , 学生 S 关系模式, 属性为学号、姓名、系、年龄和性别。
 T (Tno, Tname, Age, Sex) , 教师 T 关系模式, 属性为教师号、姓名、年龄和性别。
 C (Cno, Cname, Pcnno) , 课程 C 关系模式, 属性为课程号、课程名和先修课程号。
 SC (Sno, Cno, Grade) , 学生选课 SC 关系模式, 属性为学号、课程号和成绩。

关系模式中有下划线的属性是主码属性。图 7-16 是教学模型的一个具体的实例。

S 学生关系					T 教师关系			
<u>Sno</u>	Sname	SD	Age	Sex	<u>Tno</u>	Tname	Age	Sex
01001	贾皓昕	IS	20	男	001	方铭	34	女
01002	姚男	IS	20	男	002	章雨敬	58	男
03001	李晓红	CS	19	女	003	王平	48	女

SC 选课			C 课程关系		
<u>Sno</u>	<u>Cno</u>	Grade	<u>Cno</u>	Cname	Pcnno
01001	C001	90	C001	MS	
01001	C002	91	C002	IC	
01002	C001	95	C003	C++	C002
01002	C003	89	C004	OS	C002
03001	C001	91	C005	DBMS	C004

图 7-16 关系模型的实例

关系模型中常用的术语如下所示。

- 关系: 一个关系 R 对应一张二维表。
- 属性: 二维表中的一列称为属性, 其中 A_1, A_2, \dots, A_n 为属性名。
- 关系模式: 对关系的描述, 记为 $R(A_1, A_2, \dots, A_n)$ 。
- 元组: 二维表中一行称为元组。
- 主码: 其值能惟一地标识元组的一个或多个属性, 称为主码或关键字。
- 域: 属性 A_1, A_2, \dots, A_n 的取值范围。

- 分量：元组中的一个属性值。

关系模型与网状模型、层次模型的最大差别是用主码而不是用指针导航数据,表格简单、直观,用户只须使用简单的查询语句就可以对数据库进行操作,无须涉及存储结构和访问技术等细节。

关系模型的优点是概念单一,存储路径对用户是透明的,所以具有更好的数据独立性和安全保密性,简化了程序的开发和数据库的建立工作。正因为这一点,关系模型已经成为许多商用处理应用中的主要数据模型。

7.3 DBMS 的功能和特征

7.3.1 DBMS 的功能

DBMS 主要是对共享数据实现有效的组织、管理和存取,因此 DBMS 应具有如下几个方面的功能。

1. 数据定义

DBMS 提供数据定义语言(data definition language, DDL),用户可以对数据库的结构进行描述,包括外模式、模式和内模式的定义,数据库的完整性定义,安全保密定义,如口令、级别和存取权限等。这些定义存储在数据字典中,是 DBMS 运行的基本依据。

2. 数据库操作

DBMS 向用户提供数据操纵语言(data manipulation language, DML),实现对数据库中数据的基本操作,如检索、插入、修改和删除。DML 分为宿主型和自含型两类。所谓宿主型是指将 DML 语句嵌入某种主语言(如 C、COBOL 等)中使用。自含型是指可以单独使用的 DML 语句,供用户交互使用。

3. 数据库运行管理

数据库在运行期间对多用户环境下的并发控制、安全性检查和存取控制、完整性检查和执行、运行日志的组织管理以及事务管理和自动恢复等是 DBMS 的重要组成部分。这些功能可以保证数据库系统的正常运行。

4. 数据组织、存储和管理

DBMS 分类组织、存储和管理各种数据,包括数据字典、用户数据和存取路径等。还要确定以何种文件结构和存取方式在存储级别上组织这些数据,以提高存取效率。实现数据间的联系、数据组织和存储的基本目标是提高存储空间的利用率。

5. 数据库的建立和维护

数据库的建立和维护包括数据库的初始建立、数据的转换、数据库的转储和恢复、数

据库的重组和重构以及性能监测和分析等。

6. 其他功能

包括 DBMS 与网络中其他软件系统的通信功能,一个 DBMS 与另一个 DBMS 或文件系统的数据库转换功能等。

7.3.2 DBMS 的特征

1. DBMS 的特征

通过 DBMS 管理数据的特点如下所述。

1) 数据结构化且统一管理

数据库中的数据由 DBMS 统一管理。由于数据库系统采用复杂的数据模型表示数据结构,数据模型不仅描述数据本身的特点,还描述数据之间的联系;数据不再面向某个应用,而是面向整个应用系统;数据易维护、易扩展,数据冗余明显减少,真正实现了数据的共享。

2) 有较高的数据独立性

数据的独立性是指数据与程序独立,将数据的定义从程序中分离出去,由 DBMS 负责数据的存储,应用程序关心的只是数据的逻辑结构,无须了解数据在磁盘上的数据库中的存储形式,从而简化应用程序,大大减少应用程序编制的工作量。数据的独立性包括数据的物理独立性和数据的逻辑独立性。

3) 数据控制功能

DBMS 提供了数据控制功能,以适应共享数据的环境。数据控制功能包括对数据库中数据的安全性、完整性、并发和恢复的控制。

(1) 数据库的安全性保护:数据库的安全性(security)是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。用户只能按规定对数据进行处理,例如,划分不同的权限,有的用户只能有读数据的权限,有的用户有修改数据的权限。用户只能在规定的权限范围内操纵数据库。

(2) 数据的完整性:数据库的完整性是指数据库的正确性和相容性,是防止合法用户使用数据库时向数据库加入不符合语义的数据。保证数据库中数据是正确的,避免非法的更新。

(3) 并发控制:在多用户共享的系统中,许多用户可能同时对同一数据进行操作。并发操作带来的问题是数据的不一致性,主要有丢失更新、不可重复读和读脏数据 3 类。其主要原因是事务的并发操作破坏了事务的隔离性。DBMS 的并发控制子系统负责协调并发事务的执行,保证数据库的完整性不受破坏,避免用户得到不正确的数据。

(4) 故障恢复:数据库中的 4 类故障是事务内部故障、系统故障、介质故障及计算机

病毒。故障恢复主要是指恢复数据库本身,即在故障引起数据库当前状态不一致后,将数据库恢复到某个正确状态或一致状态。恢复的原理非常简单,就是要建立冗余(redundancy)数据。换句话说,确定数据库是否可恢复的方法就是看其包含的每一条信息是否都可以利用冗余地存储在别处的信息重构。冗余是物理级的,逻辑级通常是没有冗余的。

2. RDBS, OODBS 和 ORDBS

关系数据库系统(relation database systems, RDBS)是支持关系模型的数据库系统。在关系模型中,实体以及实体间的联系都用关系来表示。在一个给定的现实世界领域中,相应于所有实体及实体之间联系的关系的集合构成一个关系数据库,也有型和值之分。关系数据库的型也称为关系数据库模式,是对关系数据库的描述,是关系模式的集合。关系数据库的值也称为关系数据库,是关系的集合。关系数据库模式与关系数据库通常统称为关系数据库。

微机系统中的简单 DBMS 系统包括常见的 FoxBASE、FoxPro、DBASE 和 Access 等,这类 DBMS(包括网络版和单用户版)严格地讲不能算是真正的关系型数据库,它们对许多关系类型的概念并不支持,但却因为简单实用、价格低廉而拥有很大的用户市场。

面向对象的数据库系统(object-oriented database system, OODBS)是支持以对象形式进行数据建模的数据库管理系统。其支持的功能包括:支持对象的类、支持类属性的继承以及支持子类。面向对象的数据库管理系统比关系数据更优越的观念已经在一些产品中得到证实。一个面向对象的数据库系统必须符合以下两个条件:必须是一个 DBMS;必须是面向对象的,也就是说,它要与现在的面向对象语言中的定义一致,有复杂对象、对象标识、封装、对象或类、继承性、与推迟绑定结合的重载、扩展性以及计算的完整性等特点。

对象关系数据库系统(object-oriented relation database system, ORDBS)在传统的关系数据模型基础上,提供元组、数组和集合等更为丰富的数据类型以及处理新的数据类型的能力。这样形成的数据模型称为“对象关系数据模型”。基于对象关系数据模型的 DBS 称为对象关系数据库系统。

7.4 数据库系统体系结构

数据库系统是数据密集型应用的核心,其体系结构受数据库运行所在的计算机系统的影响很大,尤其是受计算机体系结构中的联网、并行和分布的影响。站在不同的角度或不同层次上看,数据库系统体系结构也不同。站在数据库管理系统的角度看,数据库系统体系结构一般采用三级模式结构。站在最终用户的角度看,数据库系统体系结构分为集

中式、分布式、客户/服务器(C/S)和并行结构。

7.4.1 数据库的三级模式结构

实际上数据库的产品很多,它们支持不同的数据模型,使用不同的数据库语言,建立在不同的操作系统上。数据的存储结构也各不相同,但体系结构基本上都具有相同的特征,采用“三级模式和两级映像”。如图 7-17 所示。

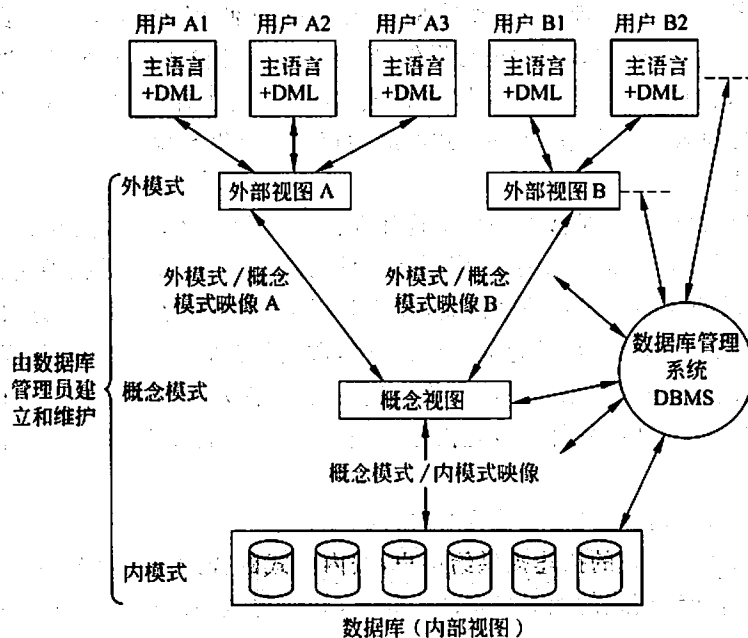


图 7-17 数据库系统体系结构

数据库系统采用的三级模式结构是数据库管理系统内部的系统结构。数据库有“行”和“值”的概念。“行”是指对某一数据的结构和属性的说明,“值”是行的一个具体赋值。

数据库系统设计员可在视图层、逻辑层和物理层对数据抽象,通过外模式、概念模式和内模式来描述不同层次上的数据特性。

1. 概念模式

概念模式也称模式是数据库中全部数据的逻辑结构和特征的描述。它由若干个概念记录类型组成,只涉及到行的描述,不涉及具体的值。概念模式的一个具体值称为模式的一个实例,同一个模式可以有很多实例。

概念模式反映的是数据库的结构及其联系,所以是相对稳定的;而实例反映的是数据库某一时刻的状态,所以是相对变动的。

需要说明的是,概念模式不仅要描述概念记录类型,还要描述记录间的联系、操作以及数据的完整性和安全性等要求。但是,概念模式不涉及存储结构和访问技术等细节。只有这样,概念模式才算做到了“物理数据独立性”。

描述概念模式的数据定义语言称为“模式 DDL”(schema data definition language)。

2. 外模式

外模式也称用户模式或子模式是用户与数据库系统的接口,是用户用到的那部分数据的描述。它由若干个外部记录类型组成。用户使用数据操纵语言 DML (data manipulation language)对数据库进行操作,实际上是对外模式的外部记录进行操作。

描述外模式的数据定义语言称为“外模式 DDL”。有了外模式后,程序员不必关心概念模式,只与外模式发生联系,按外模式的结构存储和操纵数据。

3. 内模式

内模式也称存储模式是数据物理结构和存储方式的描述,是数据在数据库内部的表示方式。定义所有的内部记录类型、索引和文件的组织方式,以及数据控制方面的细节。

例如:记录的存储方式是顺序存储,按照 B 树结构存储,还是 hash 方法存储,索引按照什么方式组织,数据是否压缩存储,是否加密,数据的存储记录结构有何规定。

需要说明的是,内部记录并不涉及物理记录,也不涉及设备的约束。比内模式更接近于物理存储和访问的那些软件机制是操作系统的一部分(即文件系统)。例如,从磁盘上读、写数据。

描述内模式的数据定义语言称为“内模式 DDL”。

总之,数据按外模式的描述提供给用户,按内模式的描述存储在磁盘上,而概念模式则提供了连接这两级模式的相对稳定的中间点,并使得这两级模式的任意一级的改变都不受另一级的牵制。

4. 两级映像

数据库系统在三级模式之间提供了两级映像:模式/内模式映像和外模式/模式映像。正因为有这两级映像才保证了数据库中的数据具有较高的逻辑独立性和物理独立性。

- 模式/内模式的映像:该映像存在于概念级和内部级之间,实现了概念模式到内模式之间的相互转换。
- 外模式/模式的映像:该映像存在于外部级和概念级之间,实现了外模式到概念模式之间的相互转换。

数据的独立性是指数据与程序独立,将数据的定义从程序中分离出去,由 DBMS 负责数据的存储,从而简化应用程序,减少应用程序编制的工作量。数据的独立性是由 DBMS 的二级映像功能来保证的。数据的独立性包括数据的物理独立性和数据的逻辑

独立性。

- 数据的物理独立性是指当数据库的内模式发生改变时,数据的逻辑结构不变。由于应用程序处理的只是数据的逻辑结构,这样物理独立性可以保证,当数据的物理结构改变了,应用程序不用改变。但是,为了保证应用程序能够正确执行,需要修改概念模式/内模式之间的映像。
- 数据的逻辑独立性指用户的应用程序与数据库的逻辑结构是相互独立的。数据的逻辑结构发生变化后,用户程序也可以不修改。但是,为了保证应用程序能够正确执行,我们需要修改外模式/概念模式之间的映像。

7.4.2 集中式数据库系统

分时系统环境下的集中式数据库系统结构诞生于 20 世纪 60 年代中期。当时的硬件和操作系统决定了分时系统环境下的集中式数据库系统结构成为早期数据库技术的首选结构。在这种系统中,不但数据是集成的,数据的管理也是集中的。数据库系统的所有功能,从形式的用户接口到 DBMS 核心都集中在 DBMS 所在的计算机上。目前大多数关系 DBMS 产品都是从这种系统结构发展起来的,这种系统现在仍然有人使用。

7.4.3 客户/服务器数据库体系结构

站在最终用户的角度看,采用客户/服务器(C/S)体系结构后,可以使某些任务在服务器上执行,另一些任务在客户机上执行。客户/服务器的一般结构如图 7-18 和图 7-19 所示。

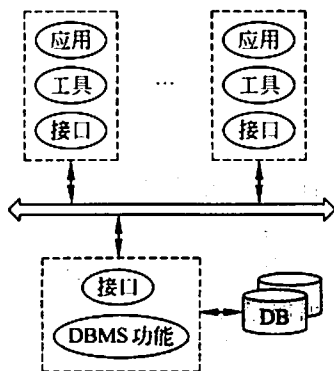


图 7-18 集中式服务器结构

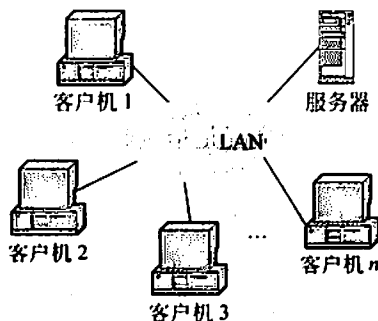


图 7-19 客户/服务器的一般结构

采用客户/服务器结构后,数据库系统功能分为前端和后端。前端主要包括图形用户界面、表格生成和报表处理等工具;后端负责存取结构、查询计算和优化、并发控制以及故障恢复等。前端与后端通过 SQL 或应用程序来接口。

采用客户/服务器体系结构的主要特点是客户机与服务器之间的职责明确,客户机主要负责数据表示服务,而服务器主要负责数据库服务。

ODBC(开放的数据库连接)和 JDBC(Java 程序数据库连接)标准定义了应用程序和数据库服务器通信的方法,也就是定义了应用程序接口。应用程序可以用其来打开与数据库的连接、发送查询和更新以及获取返回结果等。

数据库服务器一般可分为事务服务器和数据服务器。

事务服务器也称查询服务器。它提供一个接口,使得客户可以发出执行一个动作的请求。服务器响应客户请求,并将执行结果返回给客户。用户可以用 SQL、也可以通过应用程序或使用远程过程调用机制来表达请求。一个典型的事务服务器系统中有多在共享内存中访问数据的进程,包括服务器进程、锁管理进程、写进程、监视进程和检查点进程。共享内存与进程结构如图 7-20 所示。

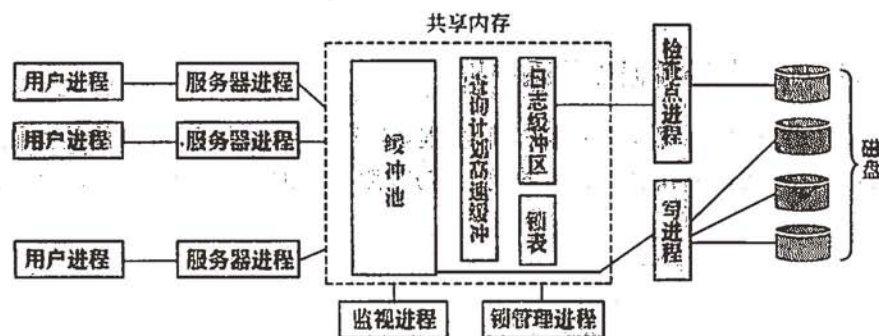


图 7-20 数据库系统中的进程

- 服务器进程是指接收用户查询、执行查询和发送回结果的进程。一个查询可以从一个用户接口上、一个运行嵌入式 SQL 的用户进程中或者经 ODBC、JDBC 等协议提交给服务器的。有的数据库系统为每个用户会话提供一个单独的进程。也有的数据库系统为所有的用户会话提供一个数据库进程,但使用多线程使多个查询并发执行。当然,也有的数据库系统使用混合结构,有多个进程,每个进程运行多个线程。
- 锁管理进程负责实现锁管理功能,包括锁授予、锁释放和死锁检测。
- 写进程负责将修改的缓冲块、日志记录写入磁盘。
- 监视进程监视其他进程。一旦发现有进程失败,将为失败的进程执行恢复操作,

例如,中止失败进程执行的所有事务,然后重新启动进程。

- 检查点进程定期执行检查点。

共享内存包含各种共享数据,如缓冲池、锁表、日志缓冲区和查询计划的高速缓存。高速缓存中的查询计划的好处是,当有同一个查询再次提交时可以重用。

数据服务器系统使得客户可以与服务器交互,以文件或页面为单位对数据进行读取或更新。数据服务器与文件服务器相比提供的功能更强,所支持的数据单位比文件还要小,如页、元组或对象。提供数据的索引机制和事务机制,使得客户机或进程发生故障时数据也不会处于不一致状态。

数据服务器系统中的客户与服务器之间具有高速的连接,客户机的处理能力与服务器相当,并且要执行的任务是计算密集型的。在这样的环境中,所有的功能都要放在客户端,数据被传送到客户机,由客户机进行所有处理,再将结果传回到服务器端。由于客户机与服务器之间通信的时间代价与本地存储器引用的代价相比要高得多,所以在实际应用中需要注意这方面的问题。

7.4.4 并行数据库系统

并行体系结构的数据库系统由多个物理上连在一起的 CPU 组成,而分布式系统的各个 CPU 在地理上是分开的。并行体系结构的数据库类型分为共享内存式多处理器和无共享式并行体系结构。

1. 共享内存式多处理器

共享内存式多处理器是指一台计算机上同时有多个活动的 CPU,它们共享单个内存和一个公共磁盘接口,如图 7-21 所示。这种并行体系结构最接近于传统的单 CPU 处理器结构,这种设计的主要挑战是用 N 个 CPU 来得到 N 倍单 CPU 的性能。但是,

因为不同的 CPU 对公共内存的访问是平等的,这样可能会导致一个 CPU 访问的数据被另一个 CPU 修改,所以必须要有特殊的处理。然而,由于内存访问采用的是一种高速机制,这种机制很难保证进行内存划分时不损失效率,所以这些共享内存访问问题会随着 CPU 个数的增加而变得难以解决。目前,IBM OS/390 也只有 12 个 CPU。

2. 无共享式并行体系结构

无共享式并行体系结构是指一台计算机上同时有多个活动的 CPU,但它们都有自己的内存和磁盘,如图 7-22 所示。其中,高速网络用粗线表示。在不产生混淆的情况下,也称为并行数据库系统。各个承担数据库服务责任的 CPU 划分它们自身的数据,通过划

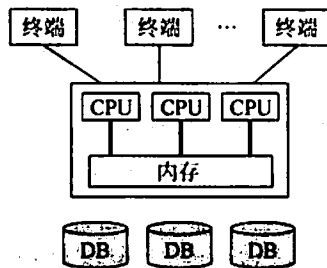


图 7-21 共享式多处理器体系结构

分的任务以及通过每秒兆位级的高速网络通信完成事务查询。

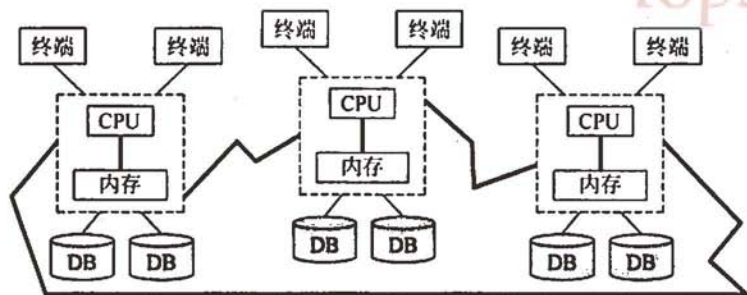


图 7-22 无共享式并行体系结构

7.4.5 分布式数据库系统

分布式 DBMS 包括物理上分布、逻辑上集中的分布式结构和物理上逻辑上都分布的分布式数据库结构两种。前者的指导思想是：把数据模式（称为全局数据模式）按数据来源和用途，合理地分布在系统的多个结点上，使大部分数据可以就地或就近存取。数据在物理上分布后，由系统统一管理，使用户感觉不到数据的分布。后者一般由两部分组成：一是本结点的数据模式，二是本结点共享的其他结点上的有关数据模式。结点间的数据共享由双方协商确定，这种数据库结构有利于数据库的集成、扩展和重新配置。

7.4.6 Web 数据库

数据库技术是计算机处理与存储数据的最有效、最成功的技术，而计算机网络的特点是资源共享，因此数据与资源共享这两种技术的结合即成为今天广泛应用的 Web 数据库（也叫网络数据库）。Web 数据库就是利用浏览器作为用户输入接口，输入所需要的数据，浏览器将这些数据传送给网站。网站再对这些数据进行处理，例如，将数据存入后台数据库，或者对后台数据库进行查询操作等。最后，网站将操作结果回传给浏览器，通过浏览器将结果告知用户。网站上的后台数据库就是 Web 数据库。

通常，Web 数据库的环境由硬件元素和软件元素组成。硬件元素包括 Web 服务器、客户机、数据库服务器和网络。如图 7-23 所示。软件元素包括客户端能够解释执行 HTML 代码的浏览器（如 IE 或 Netscape 等），Web 服务器中必须具有能自动生成 HTML 代码的程序，如 ASP、JSP 或 CGI 等，具有能自动完成数据操作指令的数据库系统，如 Access 或 SQL Server 等。互联网环境的数据库系统与 C/S 数据库体系结构非常相似。在 C/S 数据库体系结构中，需要使用服务器、网络以及一台或多台相关的 PC，只允许访问公司内部网的数据库系统。即使允许访问公司外部的数据库，客户端还需要安

装其他应用软件。

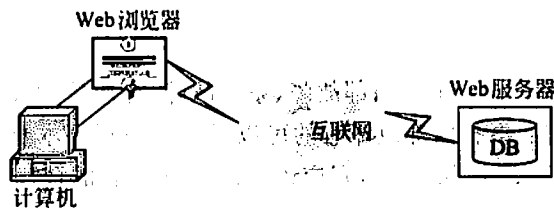


图 7-23 基于互联网环境的数据库系统

基于互联网环境的数据库体系结构是非常独特的,因为它依赖于互联网,所需的客户端软件对客户是透明的。应用软件可以只安装在一台服务器(Web 服务器)上。用户的 PC 机必须具有连接互联网的功能,并且装有 Web 浏览器。Web 浏览器通过特定的 URL 向 Web 服务器发出请求,Web 服务器收到请求后,再按照特定的请求方式访问数据库,并将数据库的执行结果反馈给用户的浏览器。最后,查询的结果通过浏览器显示在用户的 PC 上。基于互联网环境的数据库系统的最终用户应用软件的安装和维护都是非常简单的,在客户端不需要进行安装、配置和修改,这些工作只需要在服务器上完成,因此可以减少客户端和服务端软件配置不一致以及不同软件版本所带来的问题。当应用软件需要修改时,只须在 Web 服务器上进行。

7.5 数据库的控制功能

7.5.1 事务管理

事务是一个操作序列,这些操作“要么都做,要么都不做”。事务是数据库环境中不可分割的逻辑工作单位。事务和程序是两个不同的概念,一般一个程序可包含多个事务。

事务的 4 个特性是原子性(atomicity)、一致性(consistency)、隔离性(isolation)和持久性(durability)。这 4 个特性也称事务的 ACID 性质。

- 原子性:事务是原子的,要么都做,要么都不做。
- 一致性:事务执行的结果必须保证数据库从某个一致性状态变到另一个一致性状态。因此,当数据库只包含成功提交事务的结果时,称数据库处于一致性状态。
- 隔离性:事务相互隔离。当多个事务并发执行时,任一事务的更新操作,在其成功提交之前,对其他事务都是不可见的。
- 持久性:一旦事务成功提交,即使数据库崩溃,其对数据库的更新操作也将永久有效。

7.5.2 故障恢复

1. 故障的种类

数据库中存在的4类故障：事务内部故障、系统故障、介质故障及计算机病毒。

(1) 事务内部故障：事务内部故障有的可以通过事务程序本身发现。例如银行转账事务，将账户A的金额X转到账户B，则应该确保账户A新的余额等于原余额-X，账户B新的余额等于原余额+X。如果账户A的余额不足，那么，两个事务都不做，否则都做。但有些是非预期的，不能由事务程序处理，如运算溢出、并发事务发生死锁等。

(2) 系统故障：通常称为软故障，是指造成系统停止运行的任何事件，如CPU故障、操作系统故障和突然停电等。系统还常需要重新启动。

(3) 介质故障：通常称为硬故障。如磁盘损坏、磁头碰撞和瞬时强磁干扰等。此类故障发生的几率很小，但破坏性最大。

(4) 计算机病毒：是一种人为的故障和破坏，是一些恶作剧者研制的一种计算机程序，可以繁殖和传播。

2. 故障的恢复方法

故障恢复的基本原理是“建立数据冗余”(重复存储)。建立冗余数据的方法进行数据转储和登记日志文件。数据的转储分为：静态转储和动态转储、海量转储和增量转储。

1) 静态转储和动态转储

静态转储是指在转储期间不允许对数据库进行任何存取或修改操作。动态转储是在转储期间允许对数据库进行存取或修改操作，因此，动态转储和用户事务可并发执行。

2) 海量转储和增量转储

海量转储是指每次转储全部数据，增量转储是指每次只转储上次转储后更新过的数据。

3) 日志文件

在事务处理的过程中，DBMS把事务开始、事务结束以及对数据库的插入、删除和修改的每一步操作写入日志文件。一旦发生故障，DBMS的恢复子系统可以利用日志文件撤销事务对数据库的改变，回退到事务的初始状态。因此，DBMS可以利用日志文件来进行事务故障恢复和系统故障恢复，并可协助后备副本进行介质故障恢复。

事务恢复有3个步骤：

- ① 反向扫描文件日志(即从最后向前扫描日志文件)，查找该事务的更新操作。
- ② 对事务的更新操作执行逆操作。
- ③ 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样的处理，直到事务的开始标志。

3. 数据库镜像

为了避免磁盘介质出现故障而影响数据库的可用性,许多 DBMS 均提供数据库镜像功能以便用于数据库恢复。需要说明的是,数据库镜像是通过复制数据实现的。但频繁地复制数据会降低系统的运行效率。因此实际应用中往往只对关键的数据和日志文件镜像。

在 SQL 语言中,定义事务的语句有 3 条。

- BEGIN TRANSACTION 事务开始。
- COMMIT 事务提交。该操作表示事务成功地结束。它将通知事务管理器,说明该事务的所有更新操作现在可以被提交或永久地保留。
- ROLLBACK 事务回滚。该操作表示事务非成功地结束。它将通知事务管理器,说明出故障了,数据库可能处于不一致状态,该事务的所有更新操作必须回滚或撤销。

7.5.3 并发控制

所谓并发操作是指在多用户系统中,许多用户可能同时对同一数据进行操作。并发操作带来的问题是数据的不一致性,主要有 3 类:丢失更新、不可重复读和读脏数据。其主要原因是事务的并发操作破坏了事务的隔离性。DBMS 的并发控制子系统负责协调并发事务的执行,保证数据库的完整性不受破坏,避免用户得到不正确的数据。

并发控制的主要技术是封锁。基本的封锁类型有排他锁(简称 X 锁或写锁)和共享锁(简称 S 锁或读锁)。

1. 封锁

排他锁:若事务 T 对数据对象 A 加上 X 锁,则只允许 T 读取和修改 A,其他事务都不能再对 A 加任何类型的锁,直到 T 释放 A 上的锁。

共享锁:若事务 T 对数据对象 A 加上 S 锁,则只允许 T 读取 A,但不能修改 A,其他事务只能再对 A 加 S 锁,直到 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A,但在 T 释放 A 上的 S 锁之前不能对 A 进行任何修改。

2. 三级封锁协议

- 一级封锁协议:事务在修改数据 R 之前必须先对其加 X 锁,直到事务结束才释放。事务结束包括正常结束(COMMIT)和非正常结束(ROLLBACK)。一级封锁协议可以解决丢失更新问题。
- 二级封锁协议:在一级封锁协议的基础上,事务 T 在读数据 R 之前还必须先对其加 S 锁,读完后即可释放 S 锁。二级封锁协议可以解决读脏数据的问题。但由于二级封锁协议读完了数据后即可释放 S 锁,所以不能保证可重复读。

- 三级封锁协议：在一级封锁协议的基础上，事务 T 在读数据 R 之前必须先对其加 S 锁，直到事务结束时释放 S 锁。三级封锁协议除了防止丢失修改和不读“脏”数据外，还进一步防止了不可重复读。

3. 活锁与死锁

所谓活锁是指当事务 T_1 封锁了数据 R ，事务 T_2 请求封锁数据 R ，此时 T_2 只能等待。当 T_1 释放了 R 上的封锁后，如果系统首先批准了 T_2 请求，则 T_2 仍需等待。当 T_2 释放了 R 上的封锁后，又批准了 T_1 请求，……依次类推，使得 T_2 可能永远等待。这种现象就是死锁。

所谓死锁是指两个以上的事务分别请求封锁对方已经封锁的数据，导致长期等待而无法继续运行下去的现象。

4. 并发调度的可串行性

定义：多个事务的并发执行是正确的，当且仅当其结果与某一次序串行地执行它们时的结果相同，我们称这种调度策略是可串行化的调度。

可串行性是并发事务正确性的准则，按照这个准则，一个给定的并发调度，当且仅当它是可串行化的才认为是正确的调度。

5. 两段封锁协议

所谓两段封锁协议是指所有事务必须分两个阶段对数据项加锁和解锁。即把事务分为两个阶段，第一阶段是获得封锁，事务可以获得任何数据项上的任何类型的锁，但不能释放；第二阶段是释放封锁，事务可以释放任何数据项上的任何类型的锁，但不能申请。

6. 封锁的粒度

封锁对象的大小称为封锁的粒度。封锁的对象可以是逻辑单元（如属性、元组、关系、索引项、整个索引直至整个数据库），也可以是物理单元（如数据页或索引页）。

7. 事务的嵌套问题

事务是不能嵌套的，因为这违背了事务的原子性。如果事务 T_2 嵌套在事务 T_1 内，并且发生了如下事件：

```
BEGIN TRANSACTION(transaction T1);
.....
BEGIN TRANSACTION(transaction T2);
Transaction B updates tuple t;
COMMIT (transaction T2);
ROLLBACK(transaction T1);
```

如果此时将元组 t 恢复为事务 T_1 发生前的值，则事务 T_2 的 COMMIT 实际上就不是真正的 COMMIT。相反，如果保证了 T_2 的 COMMIT 的真实性，元组 t 就不能恢复为

事务 T1 发生前的值,从而 T1 的 ROLLBACK 就不是真正的 ROLLBACK。

事务不能嵌套是指当且仅当前没有事务在运行时,程序才能执行 BEGIN TRANSACTION 操作。实际上,通过放弃内层事务的持久性,可允许事务嵌套。也就是说,内层事务的 COMMIT 将提交该事务的更新操作,只限于外层事务的范围。如果外层事务以回滚操作终止,内层事务也必须回滚。从纯语义的角度看,像 SQL 这样缺少显式 BEGIN TRANSACTION 的语言,很难实现事务嵌套。因为必须有某种显式的方式指出内层事务的初始点,用以标识内层事务失败时应回滚的终点。

7.5.4 安全性和授权

除了完整性约束提供保护意外引入的不一致性之外,数据库中存储的数据还要防止未经授权的访问和恶意的破坏或修改。本节先提出数据的误用或故意使数据不一致的问题,然后给出解决这些问题的机制。

1. 安全性违例

下面是一些恶意访问的形式:

- 未经授权读取数据(窃取信息)。
- 未经授权修改数据。
- 未经授权破坏数据。

数据库安全性(data base security)意指保护数据库不受恶意访问。绝对杜绝数据库的恶意滥用是不可能的,但可以使那些企图在没有适当授权情况下访问数据库的代价足够高,以阻止绝大多数这样的访问企图。为了保护数据库,我们必须在几个层次上采取安全性措施:

- 数据库系统层次(database system)。数据库系统的某些用户获得的授权可能只允许访问数据库中有限的部分,而另外一些用户获得的授权可能允许提出查询,但不允许修改数据。保证不违反这样的授权限制是数据库系统的责任。
- 操作系统层次(operating system)。不管数据库系统多安全,操作系统安全性方面的弱点总是可能成为对数据库进行未经授权访问的一种手段。
- 网络层次(network)。由于几乎所有的数据库系统都允许通过终端或网络进行远程访问,网络软件的软件层安全性和物理安全性一样重要,不管在因特网上还是在私有的网络内。
- 物理层次(physical)。计算机系统所在的节点(一个或多个)必须在物理上受到保护,以防止入侵者强行闯入或暗中潜入。
- 人员层次(human)。对用户的授权必须格外小心,以减少授权用户接受贿赂或其他好处而给入侵者提供访问机会的可能性。

为了保证数据库安全,我们必须在上述所有层次上进行安全性防护。如果较低层次上(物理层次或人员层次)的安全性存在缺陷,高层安全性措施即使很严格也可能被绕过。下面我们将在数据库系统层次上讨论安全性。

2. 授权

我们可以赋予用户数据库各个部分上的几种形式的授权,其中包括:

- read 授权允许读取数据,但不允许修改数据。
- insert 授权允许插入新数据,但不允许修改已经存在的数据。
- update 授权允许修改数据,但不允许删除数据。
- delete 授权允许删除数据。

我们可以赋予用户上面的所有授权类型或其中一部分的组合,也可以根本不赋予任何授权。除了以上几种对数据访问的授权外,用户还可以获得修改数据库模式的授权:

- index 授权允许创建和删除索引。
- resource 授权允许创建新关系。
- alteration 授权允许添加或删除关系中的属性。
- drop 授权允许删除关系。

drop 授权和 delete 授权的区别在于 delete 授权只允许对元组进行删除。如果用户删除了关系中的所有元组,关系仍然存在,只不过是空的。如果关系被删除,那么关系就不再存在了。

可以通过 resource 授权来控制创建新关系的能力。具有 resource 授权的用户在创建新关系后自动获得该关系上的所有权限。

index 授权看起来似乎是不必要的,因为索引的创建和删除不会改变关系中的数据。事实上,索引是提高性能的一种结构。但是,索引也会消耗空间,并且所有数据库的修改都需要更新索引。如果 index 授权被授予所有用户,那么执行更新操作的用户倾向于删除索引,而提出查询的用户倾向于创建大量索引。为了使数据库管理员能够管理系统资源的使用,有必要将索引的创建作为一种权限来看待。

最大的授权是给数据库管理员的。数据库管理员可以给新用户授权,可以重构数据库,等等。这一授权类似于操作系统中提供给超级用户或操作员的权限。

3. 授权与视图

前面已经介绍了视图的概念,视图是给用户提供一个个性化数据库模型的一种手段。视图可以隐藏用户不需要看见的数据。视图隐藏数据的能力既可以简化系统的使用,又可以用于实现安全性。由于视图只允许用户关注那些感兴趣的数据,它简化了系统的使用。尽管可能不允许用户直接访问某个关系,但可能允许用户通过一个视图访问该关系的一部分。因此,关系级的安全性和视图级的安全性可以结合起来,用于限制用户只能访问所

需的数据。

在银行的例子中,考虑一个需要知道在各支行有贷款的所有客户姓名的职员。该职员不能看到与客户贷款相关的具体信息。因此,该职员对 loan 关系的直接访问必须被禁止。但是,如果该职员要访问所需信息,就必须得到对视图 cust-loan 的访问,这一视图由所有客户姓名及其贷款支行构成。此视图可以用 SQL 定义如下:

```
create view cust-loan as
  (select branch-name, customer-name
   from borrower, loan
   where borrower.loan-number=loan.loan-number)
```

假设该职员提出如下 SQL 查询:

```
select *
from cust-loan
```

显然,该职员被允许看到此查询的结果。但是,当查询处理器将此查询转换为数据库中的事实关系上的查询时,它产生的是 borrower 和 loan 上的查询。因此,系统对职员查询授权的检查必须在查询处理开始之前进行。

创建视图并不需要 resource 授权。创建视图的用户不一定能获得该视图上的所有权限,他得到的权限不会为他提供超过原有授权的其他授权。例如,在用来定义视图的关系上没有 update 授权的用户不能得到相应视图上的 update 授权。如果用户创建一个视图,而此用户在该视图上不能获得任何授权,这样的视图创建请求将被系统拒绝。在 cust-loan 的例子中,视图的创建者必须在关系 borrower 和 loan 上都具有 read 授权。

4. 权限的授予

获得某种授权的用户可能被允许将此授权传递给其他用户。但是,对于授权怎样在用户间传递我们必须格外小心,以保证这样的授权在未来的某个时候可以被收回。

作为例子,考虑银行数据库中 loan 关系上 update 权限的授予。假设最初数据库管理员将 loan 上的 update 权限授给用户 U_1 、 U_2 和 U_3 ,他们接下来又可以将定授权传递给其他用户。授权从一个用户到另一个用户的传递可以表示为授权图(authorization graph),该图的节点是用户。如果用户 U_i 将 loan 上的 update 权限授给用户 U_j ,则图中包含边 $U_i \rightarrow U_j$ 。图的根是数据库管理员。图 7-24 给出了一个示意的图。请注意用户 U_1 和 U_2 都给 U_3 授予了权限,而 U_4 只从 U_1 处获得了授权。

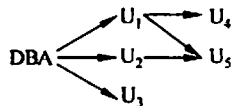


图 7-24 权限授予图

用户具有授权当且仅当存在从授权图的根(即代表数据库管理员的节点)到代表该用户节点的路径。

设数据库管理员决定收回用户 U_1 的授权。由于用户 U_4 从 U_1 处获得授权,因此其权限也应该被收回。可是,用户 U_5 既从 U_1 处又从 U_2 处获得了授权。由于数据库管理员没有从 U_2 处收回 loan 上的 update 授权, U_5 继续拥有 loan 上的 update 授权。如果 U_2 最后从 U_5 处收回授权,则 U_5 失去授权。

一对狡猾的用户可能企图通过相互授权来破坏权限回收规则,如图 7-25(a)所示。如果数据库管理员从 U_2 收回权限, U_2 保留了通过 U_3 获得的授权,如图 7-25(b)所示。如果权限接着从 U_3 处收回, U_3 似乎保留了通过 U_2 获得的授权,如图 7-25(c)所示。然而,当数据库管理员从 U_3 处收回权限时,从 U_3 到 U_2 的边以及从 U_2 到 U_3 的边就不再是从数据库管理员开始的路径的一部分了。我们要求授权图中的所有边都必须是从数据库管理员开始的路径的一部分。 U_2 和 U_3 之间的边将被删除,结果授权图如图 7-26 所示。

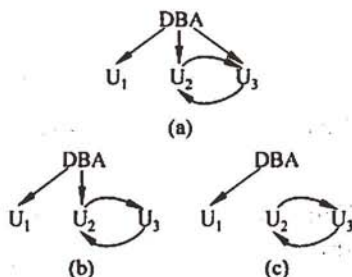


图 7-25 破坏权限回收的企图

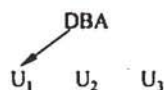


图 7-26 授权图

5. 角色

考虑一个有很多出纳的银行。每一个出纳必须对同一组关系具有同种类型的权限。无论何时指定一个新的出纳,都必须单独授予所有这些授权。

一个更好的机制是指明所有出纳应该有的授权,并单独标示出哪些数据库用户是出纳。系统可以用这两条信息来确定每一个有出纳身份的人的权限。当增加一个新出纳时,必须给他分配一个用户标识符,并且必须将他标示为一个出纳,而不需要重新单独给予出纳权限。

角色(role)的概念可用于该机制。在数据库中建立一个角色集,如同授予每一个单个用户权限一样,可将权限授予角色。分配给每个数据库用户一些他(或她)有权扮演的角色(也可能是空的)。

在我们的银行数据库里,角色的例子可以包括 teller、branch-manager、auditor 和 system-administrator。一个不是很合适的方法是建立一个 teller 用户号,允许每一个出纳用这个出纳用户号来连接数据库。该机制的问题是无法鉴别出到底哪个出纳执行了事

务,从而导致安全隐患。应用角色的好处是每个用户需要用自己的用户号连接数据库。

可以授予一个用户的任何权限都可以授予一个角色。给用户分配角色就同给用户授权一样。和其他授权一样,一个用户也可以授予为他人分配角色的权限。例如,可以授予支行经理分配出纳角色的权限。

6. 审计追踪

很多安全的数据库应用软件需要维护审计追踪(audit trail)。审计追踪是一个记录对数据库所有更改(插入/删除/更新)的日志,另外还包括一些其他信息,如哪个用户执行了更改和什么时候执行的更改等。

审计追踪在几个方面加强了安全性。例如,如果发现一个账户的余额不正确,银行也许会希望跟踪这个账户上的所有更新,以便找到错误(或欺骗性)的更新,同时也会找到执行这个更新的人。然后银行就可以利用审计追踪跟踪这些人所做的所有更新,从而找到其他错误或欺骗性的更新。

可以在关系更新操作上定义适当的触发器来建立一个审计追踪(利用标示用户名和时间的系统变量)。幸运地是,很多的数据库系统均提供了内置机制来建立审计追踪,用起来会更加方便。具体怎么建立审计追踪的细节因不同的数据库系统而不同,可以参考数据库系统用户手册来了解具体细节。

7.6 数据仓库和数据挖掘基础知识

信息技术的广泛应用将企业带入了信息爆炸的时代,管理者面对着大量等待处理的信息。这些信息分为事务型处理和信息型处理两大类。事务型处理就是通常所说的业务操作处理,是对管理信息进行的日常操作。对信息进行查询和修改,目的是为了满足不同组织特定的日常管理需要。信息型处理则是对信息做进一步的分析,为管理人员决策提供支持。这类信息的处理在现代企业中应用越来越广泛,越来越引起管理人员的重视。管理信息的信息型处理必须访问大量的历史数据才能完成,不像事务型处理那样,只对当前的信息感兴趣。

7.6.1 数据仓库

传统数据库在联机事务处理(OLTP)中获得了较大的成功,但对管理人员的决策分析要求却无法实现。因为管理人员希望对组织中的大量数据进行分析,了解组织业务的发展趋势。而传统的数据库中只能保留当前的管理信息,缺乏决策分析所需要的大量历史信息。为了满足管理人员决策分析的需要,在数据库基础上产生了能满足决策分析需要的数据环境——数据仓库(data warehouse, DW)。

数据仓库虽然是从数据库发展而来的,但是二者在许多方面有相当大的差异,如表 7-2 所示。

表 7-2 数据仓库与数据库

内 容	数 据 库	数 据 仓 库
数据内容	当前值	历史的、存档的、归纳的、计算的
数据目标	面向业务操作人员,重复处理	面向主题域、分析应用
数据特性	动态变化,按字段更新	静态、不能直接更新,只能定时添加、刷新
数据结构	高度结构化、复杂、适合操作计算	简单、适合分析
使用频率	高	中、低
数据访问量	每个事务只访问少量的记录	有的事务可能需要访问大量的记录
对响应时间的要求	以秒为单位计算	以秒、分钟、甚至小时为计算单位

1. 数据仓库的基本特性

数据仓库有这样一些重要的特性:面向主题的,数据是集成的,数据是相对稳定的,数据是反应历史变化的。

1) 面向主题的

数据仓库中的数据是按面向主题进行组织的。从信息管理的角度来看,主题就是一个较高的管理层次上对信息系统中的数据按照某一具体的管理对象进行综合、归类所形成的分析对象。从数据组织的角度来看,主题就是一些数据集合,这些数据集合对分析对象进行了比较完整的、一致的数据描述,这种数据描述不仅涉及数据自身,还涉及数据间的联系。例如,企业中的客户、产品和供应商等都可以作为主题来看待。

数据仓库的创建和使用都是围绕主题实现的,因此,必须了解如何按照决策分析来抽取主题,所抽取的主题应该包含哪些数据内容,这些数据应该如何组织。在进行主题抽取时,必须按照决策分析对象进行。例如,企业销售管理人员所关心的是本企业哪些产品销售量大、利润高?哪些客户采购的产品数量多?竞争对手的哪些产品对本企业的产品构成威胁?根据这些管理决策分析对象,就可以抽取“产品”和“客户”等主题。

2) 数据是集成的

数据仓库的集成性是指根据决策分析的要求,将分散于各处的原数据进行抽取、筛选、清理及综合等集成工作,使数据仓库中的数据具有集成性。

数据仓库所需要的数据不像业务处理系统那样直接从业务发生地获取数据。如联机事务处理系统(OLTP)、企业业务流程重组(BRP)以及基于因特网的电子商务(EC)中的数据是与业务处理联系在一起的,只为业务的日常处理服务,而不是为决策分析服务。这样,数据仓库在从业务处理系统那里获取数据时,并不能将原数据库中的数据直接加载到数据仓库中,而要进行一系列的数据预处理。即从原数据库中挑选出数据仓库所需要的

数据,然后将来自不同数据库中的数据按某一标准进行统一,如将数据源中数据的单位、字长与内容统一起来,将源数据中字段同名异义、异义同名现象消除,然后将源数据加载到数据仓库,并将数据仓库中的数据进行某种程度的综合,进行概括和聚集处理。

3) 数据是相对稳定的

数据仓库的数据主要供决策分析用,所涉及的数据操作主要是数据查询,一般情况下并不进行修改操作。数据仓库的数据反应的是一段相当长的时间内历史数据的内容,是不同时间的数据库快照的集合,以及基于这些快照进行统计、综合和重组的导出数据,而不是联机处理的数据。数据库中进行联机处理的数据经过集成输入到数据仓库中。因为数据仓库只进行数据查询操作,DBMS 中的完整性保护、并发控制,在数据仓库管理中都可以省去。但是,由于数据仓库的查询数据量往往很大,所以对数据查询提出了更高的要求,需要采用复杂的索引技术。

4) 数据是反映历史变化的

数据仓库中数据的相对稳定是针对应用来说的,数据仓库的用户进行分析处理时是不进行数据更新操作的。但并不表明在从数据集成输入数据仓库开始到最终被删除的整个数据生存周期中,数据仓库中所有的数据是永远不变的。数据仓库中的数据是反映历史变化的,这主要表现在如下 3 个方面:

① 数据仓库随时间变化不断增加新的数据内容。数据仓库系统必须不断捕捉 OLTP 数据库中变化的数据,追加到数据仓库中去。

② 数据仓库随时间变化不断删除旧的数据内容。

③ 数据仓库中包含大量的综合数据,这些数据有很多信息与时间有关,如数据经常按时间段进行综合,或隔一定的时间进行抽样等,这些数据要随时间不断地进行重新综合。

2. 数据仓库的数据模式

典型的数据仓库具有为数据分析而设计的模式,使用 OLAP 工具进行联机分析处理。因此数据通常是多维数据,包括维属性和量度属性。包含多维数据的表称为事实表,事实表通常很大。例如,一个表 sales 记录了零售商店的销售信息,其中每个元组对应一个商品售出记录,这是一个非常典型的事实表的例子。表 sales 的维包括售出的是何种商品(用商品标识表示)、商品售出的日期、商品售出的地点以及哪个顾客购买该商品等。量度属性包括售出商品的数量和金额。

为了减少存储要求,维属性通常是一些短的标识,作为参照其他表的外码。例如,事实表 sales 含有属性 item_key、time_key、branch_key 和 location_key,以及量度属性 units_sold 和 dollars_sold。其中,属性 item_key 是一个参照维表 item 的外码,表 item 含有商品名称、商品的品牌和商品所属类别等属性。属性 time_key 是一个参照维表 time

的外码,表 time 含有日、月、季和年的属性。属性 branch_key 是一个参照维表 branch 的外码,表 branch 含有出售商品的分销商的名称和分销商的类型属性。属性 location_key 是一个参照维表 location 的外码,表 location 含有销售地点的街道、城市、省份和国家等属性。由此得到一个事实表、多维表以及从事实表到多维表的参照外码的模式称为星状模式,如图 7-27 所示。

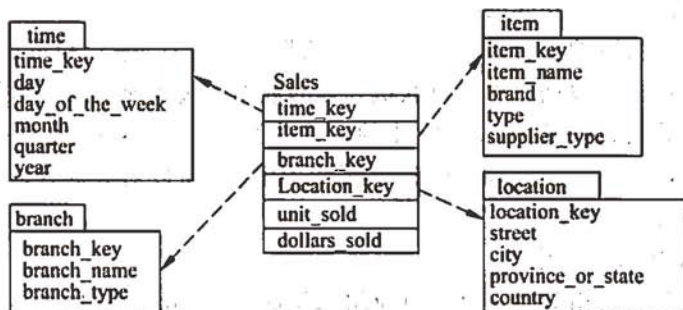


图 7-27 数据仓库的星状模式示例

更复杂的数据仓库设计可能含有多级维表,例如维表 item 含有属性 supplier_key,作为参照给出供应商细节信息的另一个维表 supplier 的外码。维表 city 含有属性 city_key,作为参照给出城市细节信息的另一个维表 supplier 的外码。这种模式称为雪花模式,如图 7-28 所示。

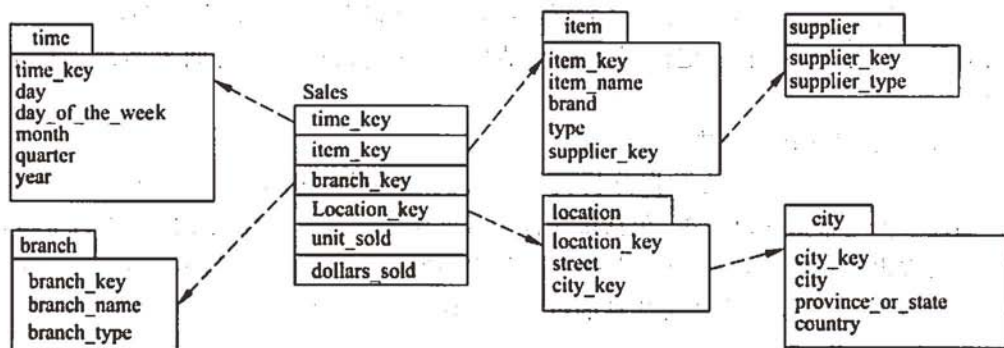


图 7-28 数据仓库的雪花模式示例

复杂的数据仓库设计可能含有不止一个事实表,图 7-29 模式中含有 Sales 和 Shipping 两个事实表,共享 location、item、time 和 branch 维表。这种模式称为事实星状模式。

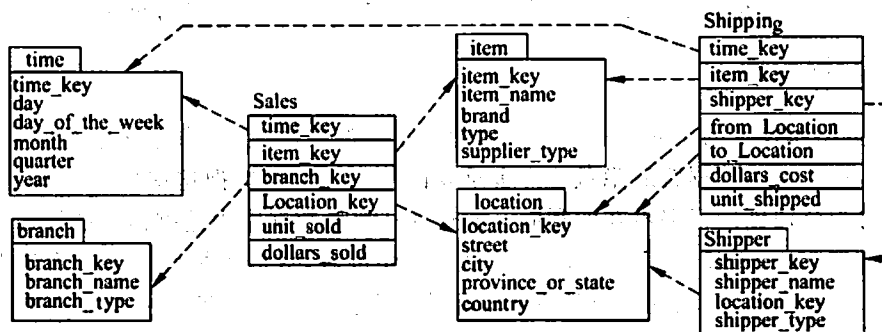


图 7-29 数据仓库的事实星状模式示例

3. 数据仓库的体系结构

数据仓库通常采用 3 层体系结构,底层为数据仓库服务器,中间层为 OLAP 服务器,顶层为前端工具。底层的数据仓库服务器一般是一个关系数据库系统,数据仓库服务器从操作型数据库或外部数据源提取数据,对数据进行清理、转换和集成等,然后装入数据仓库中。中间层的 OLAP 服务器的实现可以是关系型 OLAP,即扩充的关系型 DBMS,提供对多维数据的支持。也可以是多维的 OLAP 服务器,它是一种特殊的服务器,直接支持多维数据的存储和操作。顶层的前端工具包括查询和报表工具、分析工具及数据挖掘工具等。

从结构的角度看,可有 3 种数据仓库模型:企业仓库、数据集市和虚拟仓库。

企业仓库收集跨越整个企业各个主题的所有信息。它提供全企业范围的数据集成,数据通常都来自多个操作型数据库和外部信息提供者,并且是跨越多个功能范围的。它通常包含详细数据和汇总数据。企业数据仓库可以在传统的大型机上实现,例如 UNIX 超级服务器或并行结构平台。它需要广泛的业务建模,可能需要多年的时间来设计和建造。

数据集市包含对特定用户有用的、企业范围数据的一个子集。它的范围限于选定的主题,例如一个商场的数据集市可能限定它的主题为顾客、商品和销售。包括在数据集市中的数据通常是汇总的。通常,数据集市可以在低价格的部门服务器上实现,基于 UNIX 或 Windows NT/2000/XP。实现数据集市的周期一般是数周,而不是数月或数年。但是,如果它的规划不是企业范围的,从长远讲,可能会涉及很复杂的集成问题。根据数据的来源不同,数据集市分为独立的和依赖的两类。在独立的数据集市中,数据来自一个或多个操作型数据库或外部信息提供者,或者是一个特定部门或本地产生的数据。在依赖数据集中,数据直接来自企业数据仓库。

虚拟仓库是操作型数据库上视图的集合。为了有效地处理查询,只有一些可能的汇

总视图被物化。虚拟仓库易于建立,但需要操作型数据库服务器具有剩余能力。

7.6.2 数据挖掘

随着数据库技术的不断发展及数据库管理系统的广泛应用,数据库中存储的数据量急剧增大,在大量的数据背后隐藏着许多重要的信息。如果能把这些信息从数据库中抽取出来,将为公司创造很多潜在的利润。而这种从海量数据库中挖掘信息的技术,就称之为数据挖掘(data mining, DM)。事实上,从技术角度看,数据挖掘可以定义为从大量的、不完全的、有噪声的、模糊的以及随机的实际数据中提取隐含在其中的、人们不知道的、但又潜在有用的信息和知识的过程。

1. 数据挖掘的分类

数据挖掘工具能够对将来的趋势和行为进行预测,从而很好地支持人们的决策,比如,经过对公司整个数据库系统的分析,数据挖掘工具可以回答诸如“哪个客户对我们公司的邮件推销活动最有可能做出反应,为什么”等类似的问题。有些数据挖掘工具还能够解决一些很消耗人工时间的传统问题,因为它们能够快速浏览整个数据库,找出一些专家们不易察觉的极有用的信息。

数据挖掘技术的分类可以有多种角度。按照所挖掘的数据库种类可分为:关系型数据库的数据挖掘、数据仓库的数据挖掘、面向对象数据库的挖掘、空间数据库的挖掘、正文数据库和多媒体数据库的数据挖掘等。按所发现的知识类别可分为:关联规则、特征描述、分类分析、聚类分析以及趋势和偏差分析等。按所发现的知识抽象层次可分为:一般化知识、初级知识和多层次知识等。

数据挖掘技术是人们长期对数据库技术进行研究和开发的结果。起初各种商业数据是存储在计算机的数据库中的,然后发展到可对数据库进行查询和访问,进而发展到对数据库的即时遍历。数据挖掘使数据库技术进入了一个更高级的阶段,它不仅能对过去的数据进行查询和遍历,并且能够找出过去数据之间的潜在联系,从而促进信息的传递。现在数据挖掘技术在商业应用中已经可以马上投入使用,因为对这种技术进行支持的3种基础技术已经发展成熟。这些技术是:海量数据搜集、强大的多处理器计算机和数据挖掘算法。在数据挖掘中最常用的技术如下所述。

- 人工神经网络:仿照生理神经网络结构的非线性预测模型,通过学习进行模式识别。
- 决策树:代表着决策集的树形结构。
- 遗传算法:基于进化理论,并采用遗传结合、遗传变异以及自然选择等设计方法的优化技术。
- 近邻算法:将数据集合中每一个记录进行分类的方法。

- 规则推导：从统计意义上对数据中的“如果-那么”规则进行寻找和推导。

采用上述技术的某些专门的分析工具已经发展了大约有十年的历史,不过这些工具所面对的数据量通常较小。而现在这些技术已经被直接集成到许多大型的工业标准的数据仓库和联机分析系统中去了。将数据挖掘工具与传统数据分析工具进行比较(如表 7-3 所示),可以发现传统数据分析工具的分析重点在于向管理人员提供过去已经发生了什么,描述过去的事实。例如,上个月的销售成本是多少。而挖掘工具则在于预测未来的情况,解释过去所发生的事实的原因。例如,下个月的市场需求情况怎样,或者某个客户为什么会转向竞争对手。分析的目的也不同,前者是为了从过去的事实中列出管理人员感兴趣的事实。例如,哪些是公司最大的客户。后者则是要找出未来谁可能成为公司最大的客户。从两者分析时所需的数据量来看,也有明显的差异。前者需要的数据量并不很大,而后者需要海量数据才能运行。

表 7-3 数据挖掘工具与传统数据分析工具的比较

工具 内容	传统数据分析工具(DSS/EIS)	数据挖掘工具
工具特点	回顾型的、验证型的	预测型的、发现型的
分析重点	已发生了什么	预测未来的情况、解释发生的原因
分析目的	从最近的销售中列出最大客户	锁定未来的可能客户、减少未来的销售成本
数据集大小	数据维、维中属性数、维中数据均是少量的	数据维、维中属性数、维中数据均是庞大的
启动方式	企业管理人员、系统分析员、管理顾问启动与控制	数据与系统启动、少量的人员指导
技术情况	成熟	统计分析工具已经成熟,其他工具正在发展中

2. 数据挖掘与数据仓库的关系

根据数据挖掘的定义可以看出,数据挖掘包含一系列旨在数据库中发现有用而未发现的模式的技术,如果将其与数据仓库紧密联系在一起,将会获取意外的成功。传统的观点认为,数据挖掘技术扎根于计算科学和数学,不需要也不得益于数据仓库。这种观点并不正确,成功的数据挖掘的关键之一在于只有访问正确、完整和集成的数据,才能进行深层次的分析,寻求有益的信息,而这些正是数据仓库所能提供的。数据仓库不仅是集成数据的一种方式,数据仓库的联机分析功能 OLAP 还为数据挖掘提供了一个极佳的操作平台。如果数据仓库与数据挖掘能够实现有效的协同,将给数据挖掘带来各种便利和功能。

3. 数据挖掘技术的应用过程

数据挖掘过程一般需要经历确定挖掘对象、准备数据、建立模型、数据挖掘、结果分析

与知识应用这样几个阶段。

1) 确定挖掘对象

数据挖掘的第一步是要定义清晰的挖掘对象,认清数据挖掘的目标。数据挖掘的最后结果往往是不可预测的,但探索的问题应是有预见性的,有目标的。为了数据挖掘而挖掘数据带有盲目性,往往是不会成功的。在定义挖掘对象时,需要确定这样的问题:从何处入手?需要挖掘什么数据?要用多少数据?数据挖掘要进行到什么程度?虽然在数据挖掘中常常事先不能确定最后挖掘的结果到底是什么?例如,选择的数据是描述信用卡客户的实际支付情况,那么数据挖掘者的工作就可能是围绕着获取信用卡使用者实际支付情况而展开的。

有时还要用户提供一些先验的知识,例如概念树等。这些先验知识可能是用户业务领域知识或以前数据挖掘所获得的初步成果。这就意味着数据挖掘是一个过程,在挖掘过程中可能提出新的问题,可能尝试用其他方法来检验数据,在数据的子集上进行同样的研究。有时业务对象是一些已经理解的数据,但在某些情况下还需要对这些数据进行挖掘。此时,不是通过数据挖掘发现新的有价值的信息,而是通过数据挖掘验证假设的正确性,或者是通过同样方式的数据挖掘查看模式是否发生变化。如果在经常性的同样的数据挖掘中的一次挖掘没有出现以前同样的结果,这意味着模式已经发生了变化,可能需要进行更深层次的挖掘。例如,将数据挖掘应用于客户关系管理(CRM)中,就需要对客户关系管理的商业主题进行仔细的定义。每个CRM应用都有一个或多个商业目标,要为每个目标建立恰当的模型。例如,“提高客户对企业促销的响应率”和“提高每个客户的响应价值”这两个目标是不同的,并且在定义问题的同时,也生成了评价CRM应用结果的标准和方法,即确定了数据挖掘的评价指标。

2) 准备数据

在确定数据挖掘的业务对象后,需要搜索所有与业务对象有关的内部和外部数据,从中选出适合于数据挖掘应用的数据。对数据的选择必须在建立数据挖掘模型之前完成。选择数据后,还需要对数据进行预处理,对数据进行清洗,解决数据中的缺值、冗余、数据值的不一致性、数据定义的不一致性以及过时数据等问题。在数据挖掘时,有时还需要对数据分组,以提高数据挖掘的效率,降低模型的复杂度。

3) 建立模型

将数据转换成一个分析模型,这个分析模型是针对挖掘算法建立的。建立一个真正适合挖掘算法的分析模型,是数据挖掘的关键。

4) 数据挖掘

对所得到的经过转化的数据进行挖掘,除了完善与选择合适的算法需要人工干预外,数据挖掘工作都由数据挖掘工具自动完成。

5) 结果分析

当数据挖掘出现结果后,要对挖掘结果进行解释和评估。具体的解释和评估方法一般应根据针对数据挖掘结果所制定的决策成败来定,但是管理决策分析人员在使用数据挖掘结果之前,又希望能够对挖掘的结果进行评估,以保证数据挖掘结果在实际应用中的成功率。因此,在对数据挖掘结果进行评价时,可以考虑这样几个方面的问题:第一,建立模型相同的数据集在模型上进行操作所获得的结果要优于用不同数据集在模型上的操作结果;第二,模型的某些结果可能比其他预测结果更加准确;第三,由于模型是以样板数据为基础建立的,因此,实际结果往往会比建模时的结果差。另外,利用可视化技术可将数据挖掘结果表现的更清楚,更有利于对数据挖掘的结果分析。

6) 知识应用

数据挖掘的结果经过业务决策人员的认可,才能实际利用。要将通过数据挖掘得出的预测模式和各个领域的专家知识结合在一起,构成一个可供不同类型的人使用的应用程序。也只有通过对挖掘知识的应用,才能对数据挖掘的成果做出正确地评价。但是,在应用数据挖掘的成果时,决策人员关心的是数据挖掘的最终结果与用其他候选结果在实际应用中的差距。

数据挖掘技术可以让现有的软件和硬件更加自动化,并且可以在升级的或者新开发的平台上执行。当数据挖掘工具运行于高性能的并行处理系统上的时候,它能在数分钟内分析一个超大型的数据库。这种更快的处理速度意味着用户有更多的机会来分析数据,让分析的结果更加准确可靠,并且易于理解。数据库可以由此拓展深度和广度。在深度上,允许有更多的列存在。以往,在进行较复杂的数据分析时,专家们限于时间因素,不得不对参加运算的变量和数量加以限制,但是那些被丢弃而没有参加运算的变量有可能包含着另一些不为人知的有用信息。现在,高性能的数据挖掘工具让用户能对数据库进行通盘的深度遍历,并且把任何可能参选的变量都考虑进去,再不需要选择变量的子集来进行运算了。广度上,允许有更多的行存在。更大的样本让产生错误和变化的概率降低,这样用户就能更加精确地推导出一些虽小但颇为重要的结论。

第 8 章 关系数据库

关系数据库具有坚实的理论基础,这一理论有助于关系数据库的设计和用户对数据库信息需求的有效处理。它涉及的内容有:关系模型的基本知识、关系数据库的标准语言 SQL、查询优化以及关系数据理论。本章研究的是关系数据库,其中包括关系模型的数据结构、关系的操作和关系的完整性。

8.1 概述

建立数据库系统的主要目的是为了数据库的共享,关系数据库是目前应用得非常广泛的数据库之一,它有一套完整的理论做支持。关系模型是关系数据库的基础,由关系数据结构、关系操作集合和关系完整性规则 3 部分组成。本章介绍关系模型的基本概念、关系代数和关系演算、标准语言 SQL、查询处理策略和查询优化以及关系数据库设计理论方面的内容。

8.1.1 关系数据库的基本概念

1. 属性和域

在现实世界中,描述一个事物常常需要取其若干特征来表示。这些特征称为属性(attribute),例如学生的学号、姓名、性别、系别、年龄和籍贯等。每个属性的取值范围所对应的一个值的集合称为该属性的域(domain)。例如,学号的域是 6 位整型数,姓名的域是 10 位字符,性别的域为{男,女},……。

在关系数据模型中,对域还加了一个限制,即所有的域都应是原子数据(atomic data)。例如,整数和字符串都是原子数据,而集合、记录和数组则是非原子数据。关系数据模型的这种限制称为第一范式(first normal form,1NF)条件。但也有些关系数据模型突破了 1NF 的限制,称为非 1NF 的。

2. 笛卡儿积与关系

定义 8.1 设 D_1, D_2, \dots, D_n 为任意集合,定义 D_1, D_2, \dots, D_n 的笛卡儿积为:

$$D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_i \in D_i, i = 1, 2, \dots, n\}$$

其中每一个元素 (d_1, d_2, \dots, d_n) 叫做一个 n 元组(n -tuple 属性的个数),元组的每一个值 d_i 叫做元组一个分量。若 $D_i (i=1, 2, \dots, n)$ 为有限集,其基数(cardinal number 元组

的个数)为 $m_i (i=1, 2, \dots, n)$, 则 $D_1 \times D_2 \times \dots \times D_n$ 的基数 M 为: $M = \prod_{i=1}^n m_i$ 。笛卡儿积可以用二维表来表示。

【例 8.1】若 $D_1 = \{0, 1\}, D_2 = \{a, b\}, D_3 = \{c, d\}$, 求 $D_1 \times D_2 \times D_3$ 。

解: 根据定义, 笛卡儿积中的每一个元素应该是一个三元组, 每个分量来自不同的域, 因此结果为: $D_1 \times D_2 \times D_3 = \{(0, a, c), (0, a, d), (0, b, c), (0, b, d), (1, a, c), (1, a, d), (1, b, c), (1, b, d)\}$, 用二维表表示如图 8-1 所示。

$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} \times \begin{pmatrix} c \\ d \end{pmatrix} \Rightarrow$

D_1	D_2	D_3
0	a	c
0	a	d
0	b	c
0	b	d
1	a	c
1	a	d
1	b	c
1	b	d

图 8-1 $D_1 \times D_2 \times D_3$ 笛卡儿积的二维表表示

定义 8.2 $D_1 \times D_2 \times \dots \times D_n$ 的子集叫做在域 D_1, D_2, \dots, D_n 上的关系, 记为 $R(D_1, D_2, \dots, D_n)$, 称关系 R 为 n 元关系。

根据定义 8.2 可以得出, 一个关系也可以用二维表来表示。关系中属性的个数称为“元数”, 元组的个数称为“基数”。关系模型中的术语与一般术语的对应情况可以通过图 8-2 中的学生关系说明。图中的学生关系模式可表示为学生 (S_no, Sname, SD, Sex)。该学生关系的主码为 S_no, 属性分别为 S_no、Sname、SD 和 Sex, 属性 Sex 的域为男、女, 等等。该学生关系的元数为 4, 基数为 6。

属性 1	属性 2	属性 3	属性 4	关系模型术语	一般术语
S_no	Sname	SD	Sex	属性 关系模式	字段、数据项 记录类型
100101	张军生	通信	男	元组 1	记录 1
100102	黎晓华	通信	男	元组 2	记录 2
100103	赵敏	通信	女	元组 3	记录 3
200101	李斌斌	电子工程	男	元组 4	记录 4
300102	王莉娜	计算机	女	元组 5	记录 5
300103	吴晓明	计算机	男	元组 6	记录 6

图 8-2 学生关系与术语的对应情况

3. 关系的相关名词

- 目或度(degree): 常用 R 表示关系的名字, n 表示关系的目或度。
- 候选码(candidate key): 若关系中的某一属性或属性组的值能惟一地标识一个元组, 则称该属性或属性组为候选码。
- 主码(primary key): 若一个关系有多个候选码, 则选定其中一个为主码。
- 主属性(non-key attribute): 包含在任何候选码中的属性称为主属性。未包含在任何候选码中的属性称为非码属性。
- 外码(foreign key): 如果关系模式 R 中的属性或属性组非该关系的码, 而是其他关系的码, 那么该属性或属性组对关系模式 R 而言是外码。

例如, 在客户与贷款之间的借贷联系 $c-l(c-id, loan-no)$ 中, 属性 $c-id$ 是客户关系中的码, 所以 $c-id$ 是外码。属性 $loan-no$ 是贷款关系中的码, 所以 $loan-no$ 也是外码。

- 全码(all-key): 关系模型的所有属性组是这个关系模式的候选码, 称为全码。

例如, 关系模式 $R(T, C, S)$, 属性 T 表示教师, 属性 C 表示课程, 属性 S 表示学生。假设一个教师可以讲授多门课程, 某门课程可以由多个教师讲授, 学生可以听不同教师讲授的不同课程, 那么, 要想区分关系中的每一个元组, 这个关系模式 R 的码应为全属性 T, C 和 S , 即全码。

4. 关系的 3 种类型

(1) 基本关系(通常又称为基本表或基表): 是实际存在的表, 它是实际存储数据的逻辑表示。

(2) 查询表: 查询结果对应的表。

(3) 视图表: 是由基本表或其他视图表导出的表。由于它本身并不独立存储在数据库中, 数据库中只存放它的定义, 所以常称为虚表。

8.1.2 关系数据库模式

在数据库中要区分型和值。关系数据库中的型也称为关系数据库模式, 是关系数据库结构的描述。它包括若干域的定义以及在这些域上定义的若干关系模式。实际上, 关系的概念对应于程序设计语言中的变量的概念, 而关系模式对应于程序设计语言中的类型定义的概念。关系数据库的值是这些关系模式在某一时刻对应的关系的集合, 通常称之为关系数据库。

定义 8.3 关系的描述称为关系模式(relation schema), 可以形式化地表示为:

$$R(U, D, dom, F)$$

其中, R 表示关系名, U 是组成该关系的属性名的集合, D 是属性的域, dom 是属性向域的映像的集合, F 为属性间数据的依赖关系的集合。

通常将关系模式简记为:

$R(U)$ 或 $R(A_1, A_2, \dots, A_n)$

其中 R 为关系名, A_1, A_2, \dots, A_n 为属性名或域名, 属性向域的映像常常直接说明属性的类型和长度。通常在关系模式主属性上加下划线表示该属性为主码属性。

例如, 学生关系 S 有学号 Sno 、学生姓名 $Sname$ 、性别 Sex 、系名 SD 以及年龄 Age 等属性。课程关系 C 有课程号 Cno 、课程名 $Cname$ 和先修课程号 $PCno$ 等属性。学生选课关系 SC 有学号 Sno 、课程号 Cno 和成绩 $Grade$ 等属性。定义关系模式及主码如下(本题未考虑 F 属性间数据的依赖, 该问题在后续内容中讨论)。

(1) 学生关系模式 $S(\underline{Sno}, Sname, Sex, SD, Age)$ 。

(2) 课程关系模式 $C(\underline{Cno}, Cname, PCno)$, $Dom(PCno) = Cno$ 。这里, $PCno$ 是先行课程号, 来自 Cno 域, 但由于 $PCno$ 属性名不等于 Cno 值域名, 所以要用 Dom 来定义。但是, 不能将 $PCno$ 直接改为 Cno , 因为在关系模型中, 各列属性必须取相异的名字。

(3) 学生选课关系模式 $SC(\underline{Sno}, \underline{Cno}, Grade)$ 。 SC 关系中的 Sno 、 Cno 又分别为外码。因为它们分别是 S 、 C 关系中的主码。

8.1.3 完整性约束

完整性规则提供了一种手段, 用于保证当授权用户对数据库作修改时不会破坏数据的一致性。因此, 完整性规则防止的是对数据的意外破坏。关系模型的完整性规则是对关系的某种约束条件。关系的完整性共分为 3 类: 实体完整性、参照完整性(也称引用完整性)和用户定义完整性。

(1) 实体的完整性(entity integrity)规定基本关系 R 的主属性 A 不能取空值。

(2) 参照的完整性(referential integrity)现实世界中的实体之间往往存在某种联系。在关系模型中, 实体及实体间的联系是用关系来描述的, 这样自然就存在着关系与关系间的引用。

例如, 员工和部门的关系模式表示如下, 其中在关系模式主属性上加下划线表示该属性为主码属性。

员工(员工号, 姓名, 性别, 参加工作时间, 部门号)

部门(部门号, 名称, 电话, 负责人)

这两个关系存在着属性的引用, 员工关系中的“部门号”值必须是确实存在的部门的部门号, 即部门关系中有该部门的记录。也就是说, 员工关系中的“部门号”属性取值要参照部门关系的“部门号”属性取值。

参照完整性规定, 若 F 是基本关系 R 的外码, 它与基本关系 S 的主码 K 相对应(基本关系 R 和 S 不一定是不同的关系), 则 R 中每个元组在 F 上的值必须为:

- 或者取空值(F 的每个属性值均为空值);
- 或者等于 S 中某个元组的主码值。

(3) 用户定义的完整性(user defined integrity)就是针对某一具体关系数据库的约束条件,反映某一具体应用所涉及的数据必须满足的语义要求,由应用的环境决定。例如,银行的用户账户规定必须大于等于 100000,小于 999999。

8.2 关系运算

关系操作的特点是操作对象和操作结果都是集合。而非关系数据模型的数据操作方式则采用一次一个记录的方式。关系数据语言分为 3 类:关系代数语言、关系演算语言和具有关系代数和关系演算双重特点的语言(如 SQL)。关系演算语言包含元组关系演算语言(如 Alpha, Quel)和域关系演算语言(如 QBE)。

关系代数语言、元组关系演算和域关系演算是抽象查询语言,它与具体的 DBMS 中实现的实际语言并不一样,但是可以用它评估实际系统中查询语言能力的标准。

8.2.1 关系代数运算

1. 关系代数运算的基本运算符

关系代数运算符有 4 类:集合运算符、专门的关系运算符、算术比较符和逻辑运算符。根据运算符的不同,关系代数运算可分为传统的集合运算和专门的关系运算。传统的集合运算是从关系的水平方向进行的,包括并、交、差及广义笛卡儿积。专门的关系运算既可以从关系的水平方向进行运算,又可以按关系的垂直方向运算,包括选择、投影、连接以及除法。如表 8-1 所示。

在表 8-1 中,并、差、笛卡儿积、投影和选择是 5 种基本的运算,因为其他运算都可以通过基本运算导出。

1) 并(union)

关系 R 与 S 具有相同的模式,即 R 与 S 的元数相同(结构相同)。关系 R 与 S 的并是由属于 R 和属于 S 的元组构成的集合,记作 $R \cup S$ 。其形式定义如下:

$$R \cup S = \{t \mid t \in R \vee t \in S\}$$

式中 t 为元组变量。

2) 差(difference)

关系 R 与 S 具有相同的模式,关系 R 与 S 的差是由属于 R 但不属于 S 的元组构成的集合,记作 $R - S$ 。其形式定义如下:

$$R - S = \{t \mid t \in R \wedge t \notin S\}$$

表 8-1 关系代数运算符

运 算 符		含 义	运 算 符		含 义
集合运算符	\cup	并 差 交 笛卡儿积	比较运算符	$>$	大于
	$-$			\geq	大于等于
	\cap			$<$	小于
	\times			\leq	小于等于
				$=$	等于
				\neq	不等于
专门的关系运算符	σ	选择 投影 连接 除	逻辑运算符	\neg	非
	π			\wedge	与
	\bowtie			\vee	或
	\div				

3) 广义笛卡儿积(extended cartesian product)

两个元数分别为 n 目和 m 目的关系 R 和 S 的广义笛卡儿积是一个 $(n+m)$ 列的元组的集合。元组的前 n 列是关系 R 的一个元组,后 m 列是关系 S 的一个元组,记作 $R \times S$ 。其形式定义如下:

$$R \times S = \{t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S\}$$

如果 R 和 S 中有相同的属性名,可在属性名前加关系名作为限定,以示区别。若 R 有 K_1 个元组, S 有 K_2 个元组,则 R 和 S 的广义笛卡儿积有 $K_1 \times K_2$ 个元组。

注意:在本教材中的序偶 $\langle t^n, t^m \rangle$ 表示元组 t^n 和 t^m 拼接成的一个元组。

4) 投影(projection)

投影运算是从关系的垂直方向进行运算,即在关系 R 中选择出若干属性列 A 组成新的关系,记作 $\pi_A(R)$ 。其形式定义如下:

$$\pi_A(R) = \{t[A] \mid t \in R\}$$

5) 选择(selection)

选择运算是从关系的水平方向进行运算,是从关系 R 中选择满足给定条件的诸元组,记作 $\sigma_F(R)$ 。其形式定义如下:

$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = \text{True}\}$$

其中, F 中的运算对象是属性名(或列的序号)或常数,运算符为算术比较符($<$, \leq , $>$, \geq , $=$, \neq)和逻辑运算符(\wedge , \vee , \neg)。例如, $\sigma_{a_1 \geq 8}(R)$ 表示选取 R 关系中第一个属性值大于等于第六个属性值的元组; $\sigma_{a_1 > '6'}(R)$ 表示选取 R 关系中第一个属性值大于等于‘6’的元组。

【例 8.2】设有关系 R 、 S 如图 8-3 所示,请求出 $R \cup S$ 、 $R - S$ 、 $R \times S$ 、 $\pi_{A,C}(R)$ 、

$\sigma_{A>B}(R)$ 和 $\sigma_{3<4}(R \times S)$ 。

R 关系			S 关系		
A	B	C	A	B	C
a	b	c	b	a	d
b	a	d	d	f	g
c	d	e	f	h	k
d	f	g			

图 8-3 关系 R 和 S

解: $R \cup S$ 、 $R - S$ 、 $R \times S$ 、 $\pi_{A,C}(R)$ 、 $\sigma_{A>B}(R)$ 和 $\sigma_{3<4}(R \times S)$ 结果如图 8-4 所示。其中, $R \times S$ 后生成的关系属性名有重复, 按照关系“属性不能重名”的性质, 通常采用“关系名. 属性名”的格式。 $\sigma_{3<4}(R \times S)$ 的含义是 $R \times S$ 后“选取第三个属性值小于第四个属性值”的元组。由于 $R \times S$ 的第三个属性为 $R.C$, 第四个属性是 $S.A$, 因此 $\sigma_{3<4}(R \times S)$ 的含义也是 $R \times S$ 后“选取 $R.C$ 值小于 $S.A$ 值”的元组。

R ∪ S			R × S					
A	B	C	R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	a	b	c	b	a	d
b	a	d	a	b	c	d	f	g
c	d	e	a	b	c	f	h	k
d	f	g	b	a	d	b	a	d
f	h	k	b	a	d	d	f	g
			b	a	d	f	h	k
			c	d	e	b	a	d
			c	d	e	d	f	g
			c	d	e	f	h	k
			d	f	g	b	a	d
			d	f	g	d	f	g
			d	f	g	f	h	k

R - S			$\sigma_{3<4}(R \times S)$					
A	B	C	R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	a	b	c	d	f	g
c	d	e	b	a	d	f	h	k

$\pi_{A,C}(R)$								
A		C						
a		c						
b		d						
c		e						
d		g						

$\sigma_{A>B}(R)$								
A	B	C						
b	a	d						

图 8-4 运算结果

2. 扩展的关系运算符

扩展的关系运算可以从基本关系运算中导出。主要包括选择、投影、连接、除法、广义笛卡儿积和外连接等。

1) 交(intersection)

关系 R 与 S 具有相同的模式, 关系 R 与 S 的交是由属于 R 同时又属于 S 的元组构成的集合, 记作 $R \cap S$ 。其形式定义如下:

$$R \cap S = \{t \mid t \in R \wedge t \in S\}$$

显然, $R \cap S = R - (R - S)$, 或者 $R \cap S = S - (S - R)$ 。

2) 连接(join)

连接分为 θ 连接、等值连接及自然连接 3 种。连接运算是从两个关系 R 和 S 的笛卡儿积中选取满足条件的元组。因此, 可以认为笛卡儿积是无条件连接, 其他连接操作是有条件连接。

(1) θ 连接: 从 R 与 S 的笛卡儿积中选取属性间满足一定条件的元组。记作:

$$R \bowtie_{X \theta Y} S = \{t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[X] \theta t^m[Y]\}$$

其中: ' $X \theta Y$ ' 为连接的条件, θ 是比较运算符, X 和 Y 分别为 R 和 S 上度数相等, 且可比的属性组。 $t^n[X]$ 表示 R 中 t^n 元组相应于属性 X 的一个分量。 $t^m[Y]$ 表示 S 中 t^m 元组相应于属性 Y 的一个分量。需要说明的是:

① θ 连接也可以表示为

$$R \bowtie_{i \theta j} S = \{t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[i] \theta t^m[j]\}$$

其中: $i=1, 2, \dots, n, j=1, 2, \dots, m$, ' $i \theta j$ ' 的含义为从两个关系 R 和 S 中选取 R 的第 i 列和 S 的第 j 列之间满足 θ 运算的元组进行连接。

② θ 连接可以由基本的关系运算笛卡儿积和选取运算导出。因此 θ 连接可表示为

$$R \bowtie_{X \theta Y} S = \sigma_{X \theta Y}(R \times S) \quad \text{或} \quad R \bowtie_{i \theta j} S = \sigma_{i \theta (i+j)}(R \times S)$$

【例 8.3】 设有关系 R 、 S 如图 8-3 所示, 求 $R \bowtie_{R.A < S.B} S$ 。

解: 本题的连接条件为 $R.A < S.B$, 意为将 R 关系中属性 A 的值小于 S 关系中属性 B 的值的元组取出来作为结果集的元组。结果集为 $R \times S$ 后选出满足条件的元组, 并且结果集的属性为 $R.A, R.B, R.C, S.A, S.B$ 和 $S.C$ 。结果如图 8-5 所示。

(2) 等值连接: 当 θ 为“=”时, 称之为等值连接, 记为 $R \bowtie_{X=Y} S$ 。其形式定义如下:

$$R \bowtie_{X=Y} S = \{t \mid t = \langle t^n, t^m \rangle \wedge t^n \in R \wedge t^m \in S \wedge t^n[X] = t^m[Y]\}$$

(3) 自然连接: 是一种特殊的等值连接, 它要求两个关系中进行比较的分量必须是相同的属性组, 并且在结果集中将重复属性列去掉。若 t^n 表示 R 关系的元组变量, t^m 表示 S 关系的元组变量, R 和 S 具有相同的属性组 B , 且 $B = (B_1, B_2, \dots, B_K)$ 。假定 R 关系

R.A	R.B	R.C	S.A	S.B	S.C
a	b	c	d	f	g
a	b	c	f	h	k
b	a	d	d	f	g
b	a	d	f	h	k
c	d	e	d	f	g
c	d	e	f	h	k
d	f	g	d	f	g
d	f	g	f	h	k

图 8-5 $R \bowtie S$

的属性为 $A_1, A_2, \dots, A_{n-k}, B_1, B_2, \dots, B_K$, S 关系的属性为 $B_1, B_2, \dots, B_K, B_{K+1}, B_{K+2}, \dots, B_m$ 。为 S 的元组变量 t^m 去掉重复属性 B 所组成的新的元组变量为 t^{m*} , 可以记为 $R \bowtie S$ 。其形式定义如下:

$$R \bowtie S = \{t \mid t = \langle t^n, t^{m*} \rangle \wedge t^n \in R \wedge t^{m*} \in S \wedge R.B_1 = S.B_1 \wedge R.B_2 = S.B_2 \wedge \dots \wedge R.B_n = S.B_n\}$$

自然连接可以由基本的关系运算笛卡儿积和选取运算导出, 因此自然连接可表示为:

$$R \bowtie S = \prod_{A_1, A_2, \dots, A_{n-k}, R.B_1, R.B_2, \dots, R.B_K, B_{K+1}, B_{K+2}, \dots, B_m} (\sigma_{R.B_1 = S.B_1 \wedge R.B_2 = S.B_2 \wedge \dots \wedge R.B_K = S.B_K} (R \times S))$$

特别需要说明的是: 一般连接是从关系的水平方向运算, 而自然连接不仅要关系的水平方向, 而且要从关系的垂直方向运算。因为自然连接要去掉重复属性, 如果没有重复属性, 那么自然连接就变成笛卡儿积。

【例 8.4】设有关系 R 、 S 如图 8-6 所示, 求 $R \bowtie S$ 。

A	B	C
a	b	c
b	a	d
c	d	e
d	f	g

(a) 关系 R

A	C	D
a	c	d
d	f	g
b	d	g

(b) 关系 S

图 8-6 关系 R 、 S

解: 本题要求 R 与 S 关系的自然连接。自然连接是一种特殊的等值连接, 它要求在两个关系中进行比较的分量必须是相同的属性组, 并且在结果中将重复属性列去掉。本题 R 与 S 关系中相同的属性组为 AC , 因此, 结果集中的属性列应为 $ABCD$ 。其结果如图

8-7 所示。

A	B	C	D
a	b	c	d
b	a	d	g

图 8-7 $R \bowtie S$

3) 除(division)

除运算是同时从关系的水平方向和垂直方向进行运算。给定关系 $R(X,Y)$ 和 $S(Y,Z)$, X,Y,Z 为属性组。 $R \div S$ 应当满足元组在 X 上的分量值 x 的象集 Y_x 包含关系 S 在属性组 Y 上投影的集合。其形式定义如下:

$$R \div S = \{t^n[X] \mid t^n \in R \wedge \pi_Y(S) \subseteq Y_x\}$$

其中: Y_x 为 x 在 R 中的象集, $x = t^n[X]$ 。且 $R \div S$ 的结果集的属性组为 X 。

【例 8.5】 设有关系 R, S 如图 8-8 所示, 求 $R \div S$ 。

A	B	C	D
a	b	c	d
a	b	e	f
a	b	h	k
b	d	e	f
b	d	d	l
c	k	c	d
c	k	e	f

(a) R

C	D
c	d
e	f

(b) S

A	B
a	b
c	k

(c) $R \div S$

图 8-8 $R \div S$

解: 根据除法定义, 此题的 X 为属性 AB , Y 为属性 CD 。 $R \div S$ 应当满足元组在属性 AB 上的分量值 x 的象集 Y_x 包含关系 S 在 CD 上投影的集合。

关系 S 在 Y 上的投影为 $\pi_{CD}(S) = \{(c,d), (e,f)\}$ 。对于关系 R , 属性组 X (即 AB) 可以取 3 个值 $\{(a,b), (b,d), (c,k)\}$, 它们的象集分别为:

$$\text{象集 } CD_{(a,b)} = \{(c,d), (e,f), (h,k)\}$$

$$\text{象集 } CD_{(b,d)} = \{(e,f), (d,l)\}$$

$$\text{象集 } CD_{(c,k)} = \{(c,d), (e,f)\}$$

由于上述象集包含 $\pi_{CD}(S)$ 的有 (a,b) 和 (c,k) , 所以, $R \div S = \{(a,b), (c,k)\}$, 结果如图 8-8(c) 所示。

【例 8.6】 设学生课程数据库中有学生 S 、课程 C 和学生选课 SC 3 个关系, 如图 8-9 所示。请用关系代数表达式表达如下检索问题。

- (1) 检索选修课程名为“数学”的学生号和姓名。
- (2) 检索至少选修了课程号为‘1’和‘3’的学生号。
- (3) 检索选修了‘操作系统’或‘数据库’课程的学号和姓名。
- (4) 检索年龄在 18 到 20 之间(含 18 和 20)的女生的学号、姓名及年龄。
- (5) 检索选修了“数据库”课程的学生的学号、姓名及成绩。
- (6) 检索选修全部课程的学生姓名及所在系。
- (7) 检索选修课程包括“1042”号学生所学课程的学生学号。
- (8) 检索不选修 2 号课程的学生姓名和所在系。

Sno	Sname	Sex	SD	Age
3001	王 平	女	计算机	18
3002	张 勇	男	计算机	19
4003	黎 明	女	机 械	18
4004	刘明远	男	机 械	19
1041	赵国庆	男	通 信	20
1042	樊建玺	男	通 信	20

S

Cno	Cname	PCno	Credit
1	数据库	3	3
2	数 学		4
3	操作系统	4	4
4	数据结构	7	3
5	数字通信	6	3
6	信息系统	1	4
7	程序设计	2	2

C

Sno	Cno	Grade
3001	1	93
3001	2	84
3001	3	84
3002	2	83
3002	3	93
1042	1	84
1042	2	82

SC

图 8-9 S、C、SC 关系

解：(1) 检索选修课程名为“数学”的学生号和姓名的关系代数表达式如下：

$$\pi_{Sno, Sname}(\sigma_{Cname='数学'}(S \bowtie SC \bowtie C)) \quad \text{或} \quad \pi_{1,2}(\sigma_{8='数学'}(S \bowtie SC \bowtie C))$$

上述表达式 $S \bowtie SC \bowtie C$ 自然连接后重复的属性列为学号 Sno 和课程号 Cno, 去掉重复属性列的结果如图 8-10 所示。从图中可见, 满足课程名为“数学”的只有三个元组, 对 Sno 和 Cno 投影的结果如图 8-11 所示。由于 Sno、Cno 和 Cname 分别对应第 1、2 和 8 列属性, 所以上述表达式还可以写为: $\pi_{1,2}(\sigma_{8='数学'}(S \bowtie SC \bowtie C))$ 。

Sno	Sname	Sex	SD	Age	Cno	Grade	Cname	PCno	Credit
3001	王 平	女	计算机	18	1	93	数据库	3	3
3001	王 平	女	计算机	18	2	84	数 学		4
3001	王 平	女	计算机	18	3	84	操作系统	4	4
3002	张 勇	男	计算机	19	2	83	数 学		4
3002	张 勇	男	计算机	19	3	93	操作系统	4	4
1042	樊建玺	男	通 信	20	1	84	数据库	3	3
1042	樊建玺	男	通 信	20	2	82	数 学		4

图 8-10 $S \bowtie SC \bowtie C$

(2) 检索至少选修了课程号为‘1’和‘3’的学生号有如下两种解题思路。

第一种：关系代数表达式为 $\pi_1(\sigma_{1=4 \wedge 2='1' \wedge 5='3'}(SC \times SC))$ 。若设 $SC \times SC$ 中的第一个 SC 关系为 $S1$ ，与第二个 SC 关系为 $S2$ ，那么，该关系表达式的含义为先从 $SC \times SC$ 中选取满足条件 $S1.Sno = S2.Sno$ 、 $S1.Cno = '1'$ 和 $S2.Cno = '3'$ 的元组，最后投影第一个属性列 Sno 即为所求结果集。

Sno	Sname
3001	王 平
3002	张 勇
1042	樊建玺

图 8-11 对 Sno 和 Cno 的投影结果

第二种：关系代数表达式为 $\pi_{Sno, Cno}(SC) \div \pi_{Cno}(\sigma_{Cno='1' \vee Cno='3'}(C))$ 。

- 表达式 $\pi_{Cno}(\sigma_{Cno='1' \vee Cno='3'}(C))$ 就是构造一个临时关系 K ，其属性为 Cno ，结果如下：

$$K = \pi_{Cno}(\sigma_{Cno='1' \vee Cno='3'}(C)) = \{1, 3\}$$

- 查询表达式 $\pi_{Sno, Cno}(SC) \div K$ 的结果集的学生号所选的课程号应包括 K 。

求解的过程就是对 $\pi_{Sno, Cno}(SC)$ 的每一个元组逐一求某一学生的象集。因为所求的是 Sno ，所以 X 为 Sno ， Y 为 Cno ，象集为 Cno_{Sno} 。将 Sno 的值逐一代入求象集：

$$Cno_{3001} = \{1, 2, 3\}$$

$$Cno_{3002} = \{2, 3\}$$

$$Cno_{1042} = \{1, 2\}$$

从上可以看出，只有 3001 包含了 K 在 Cno 的投影，所以， $\pi_{Sno, Cno}(SC) \div K = \{3001\}$ 。

(3) 检索选修了‘操作系统’或‘数据库’课程的学号和姓名的关系代数表达式为：

$$\pi_{Sno, Sname}(S \bowtie (\sigma_{Cname='操作系统' \vee Cname='数据库'}(SC \bowtie C)))$$

(4) 检索年龄在 18 到 20 之间(含 18 和 20)的女生的学号、姓名及年龄的关系代数表达式为：

$$\pi_{Sno, Sname, Age}(\sigma_{Age \leq '18' \wedge Age \geq '20'}(S))$$

(5) 检索选修了“数据库”课程的学生学号、姓名及成绩的关系代数表达式为：

$$\pi_{Sno, Sname, Grade}(\sigma_{Cname='数据库'}(S \bowtie SC \bowtie C))$$

(6) 检索选修全部课程的学生姓名及所在系的关系代数表达式为:

$$\pi_{Sname, SD}(S \bowtie (\pi_{Sno, Cno}(SC) \div \pi_{Cno}(C)))$$

对于本题给出的具体关系,求解过程分析如下:

- 表示全部课程的临时关系 $K = \pi_{Cno}(C) = \{1, 2, 3, 4, 5, 6, 7\}$ 。
- 查询选修了所有课程的学生号为 $\pi_{Sno, Cno}(SC) \div K = \{\phi\}$, 因为学生所选课程分别为 $Cno_{3001} = \{1, 2, 3\}$ 、 $Cno_{3002} = \{2, 3\}$ 和 $Cno_{1042} = \{1, 2\}$, 所以 3001、3002 和 1042 都没有包含 K , 故结果集为空。
- 与 S 关系进行自然连接, 再对学生姓名 $Sname$ 和学号 SD 投影的结果也为空。

(7) 检索选修课程包含“1042”号学生所学课程的学生学号的关系表达式如下:

$$\pi_{Sno, Cno}(SC) \div \pi_{Cno}(\sigma_{Sno='1042'}(SC))$$

(8) 检索不选修 2 号课程的学生姓名和所在系的关系代数表达式为:

$$\pi_{Sname, SD}(SC) - \pi_{Sname, SD}(\sigma_{Cno='2'}(S \times SC))$$

上式也可以用属性列号替换属性名, 将上式写成如下等价的关系代数表达式:

$$\pi_{2,4}(SC) - \pi_{2,4}(\sigma_{6='2'}(S \bowtie SC))$$

4) 广义投影(generalized projection)

广义投影运算允许在投影列表中使用算术运算, 实现了对投影运算的扩充。

若有关系 R , 条件 F_1, F_2, \dots, F_n 中的每一个都是涉及 R 中常量和属性的算术表达式, 那么广义投影运算的形式定义为: $\pi_{F_1, F_2, \dots, F_n}(R)$ 。

【例 8.7】 设有信贷额度关系模式 $credit-in(C_name, limit, Credit_balance)$, 其中的属性分别表示用户姓名、信贷额度和到目前为止的花费。图 8-12(a) 表示了一个具体的关系 $credit-in$ 。若要查询每个用户还能花费多少, 可以用关系代数表达式 $\pi_{C_name, limit-credit_balance}(credit-in)$ 来表示, 查询结果如图 8-12(b) 所示。

C_name	limit	Credit_balance
王伟峰	2500	1800
吴 桢	3100	2000
黎建明	2380	2100
刘 柯	5600	3600
徐国平	8100	5800
景莉红	6000	4500

(a) $credit-in$

C_name	Limit-Credit_balance
王伟峰	700
吴 桢	1100
黎建明	280
刘 柯	2000
徐国平	2300
景莉红	1500

(b) $\pi_{C_name, limit-credit_balance}(credit-in)$

图 8-12 信贷额度关系

5) 外连接(outer join)

外连接运算是连接运算的扩展,可以处理缺失的信息。对于图 8-9 的 S 和 SC 关系,当我们对其进行自然连接 $S \bowtie SC$ 时,其结果如图 8-13 所示。

Sno	Sname	Sex	SD	Age	Cno	Grade
3001	王 平	女	计算机	18	1	93
3001	王 平	女	计算机	18	2	84
3001	王 平	女	计算机	18	3	84
3002	张 勇	男	计算机	19	2	83
3002	张 勇	男	计算机	19	3	93
1042	樊建玺	男	通 信	20	1	84
1042	樊建玺	男	通 信	20	2	82

图 8-13 $S \bowtie SC$

从图 8-13 可以看出 S 与 SC 自然连接 $S \bowtie SC$ 的结果丢失了黎明、刘明远和赵国庆的相关信息。如果使用外连接,我们可以避免这样的信息丢失。外连接运算有三种:左外连接、右外连接和全外连接。

(1) 左外连接(left outer join) $\bowtie\leftarrow$

左外连接取出左侧关系中所有与右侧关系中任一元组都不匹配的元组,用空值 NULL 填充所有来自右侧关系的属性,构成新的元组,将其加入自然连接的结果中。对于图 8-9 的 S 和 SC 关系,当我们对其进行左外连接 $S \bowtie\leftarrow SC$ 时,其结果如图 8-14 所示。

Sno	Sname	Sex	SD	Age	Cno	Grade
3001	王 平	女	计算机	18	1	93
3001	王 平	女	计算机	18	2	84
3001	王 平	女	计算机	18	3	84
3002	张 勇	男	计算机	19	2	83
3002	张 勇	男	计算机	19	3	93
4003	黎 明	女	机 械	18	NULL	NULL
4004	刘明远	男	机 械	19	NULL	NULL
1041	赵国庆	男	通 信	20	NULL	NULL
1042	樊建玺	男	通 信	20	1	84
1042	樊建玺	男	通 信	20	2	82

图 8-14 $S \bowtie\leftarrow SC$

(2) 右外连接(right outer join) $\rightarrow\bowtie$

右外连接取出右侧关系中所有与左侧关系中任一元组都不匹配的元组,用空值

NULL 填充所有来自左侧关系的属性,构成新的元组,将其加入自然连接的结果中。对于图 8-9 的 S 和 SC 关系,当我们对其进行右外连接 $SC \bowtie_r C$ 时,其结果如图 8-15 所示。

Sno	Cno	Grade	Cname	PCno	Credit
3001	1	93	数据库	3	3
3001	2	84	数 学		4
3001	3	84	操作系统	4	4
3002	2	83	数 学		4
3002	3	93	操作系统	4	4
1042	1	84	数据库	3	3
1042	2	82	数 学		4
NULL	4	NULL	数据结构	7	3
NULL	5	NULL	数字通信	6	3
NULL	6	NULL	信息系统	1	4
NULL	7	NULL	程序设计	2	2

图 8-15 $SC \bowtie_r C$

(3) 全外连接(full outer join) \bowtie_{full}

全外连接完成左外连接和右外连接的操作。即填充左侧关系中所有与右侧关系中任一元组都不匹配的元组,再填充右侧关系中所有与左侧关系中任一元组都不匹配的元组,将产生的新元组加入自然连接的结果中。

【例 8.8】 设有关系 R、S 如图 8-16 所示,求 $R \bowtie S$ 、 $R \bowtie_r S$ 和 $R \bowtie_{full} S$ 。

A	B	C
a	b	c
b	a	d
c	d	e
d	f	g

(a) 关系 R

B	C	D
b	c	d
d	e	g
f	d	g
d	e	c

(b) 关系 S

图 8-16 关系 R、S

对于图 8-16 的 R、S 关系,在对其进行左外连接 $R \bowtie_l S$,右外连接 $R \bowtie_r S$,全外连接 $R \bowtie_{full} S$ 时,其结果分别如图 8-17(a)、(b)和(c)所示。

6) 聚集函数

聚集运算是关系代数运算中的一个非常重要的扩展。聚集函数输入一个值的集合,返回单一值作为结果。例如,集合 $\{2, 4, 6, 8, 10, 15\}$ 。将聚集函数 sum 用于该集合时返回和 45。将聚集函数 avg 用于该集合时返回平均值 7.5。将聚集函数 count 用于该集合

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
b	a	d	NULL
d	f	g	NULL

(a) 左外连接 $R \bowtie S$

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
NULL	f	d	g

(b) 右外连接 $R \bowtie S$

A	B	C	D
a	b	c	d
c	d	e	g
c	d	e	c
b	a	d	NULL
d	f	g	NULL
NULL	f	d	g

(c) 全外连接 $R \bowtie S$

图 8-17 $R \bowtie S, R \bowtie S, R \bowtie S$

时返回集合中元数的个数 6。将聚集函数 min 用于该集合时返回最小值 2。将聚集函数 max 用于该集合时返回最大值 15。

需要说明的是,使用聚集函数的集合中,一个值可以出现多次,值出现的顺序是无关紧要的,这样的集合称之为多重集。集合是多重集的一个特例,其中每个值都只出现一次。

但有时在计算聚集函数前必须去掉重复值,此时可以将“distinct”用连接符附加在函数名后,如 count-distinct。

【例 8.9】 学生选课关系 SC 如图 8-18 所示,请求出选课关系中出现的课程数,选学每门课程的学生数,每门课程的平均成绩及最高成绩。

Sno	Cno	Grade
3001	1	93
3001	2	84
3001	3	84
3002	2	83
3002	3	93
2010	1	86
1042	1	84
1042	2	82

图 8-18 SC

解:对于“选课关系中出现的课程数”,每门课程只应计算一次,而不管有多少学生选该门课程。其查询表达式为 $\text{count-distinct}_{Cno}(SC)$ 。

对于“选学每门课程的学生数”应该按课程号分组,然后组内统计学生数。其查询表达式为 $\text{count}_{Sno}(SC)$ 。聚集运算符 G 左侧的属性 Cno 表示输入关系 SC 必须按照 Cno 的值进行分组,结果如图 8-19(a) 所示。然后统计各个组中的学生数,统计的结果如图 8-19(b)所示。

Sno	Cno	Grade
3001	1	93
2010	1	86
1042	1	84
3001	2	84
3002	2	83
1042	2	82
3001	3	84
3002	3	93

(a)

Cno	Student-num
1	3
2	3
3	2

(b)

Cno	avg of Grade	max of Grade
1	87.7	93
2	83	84
3	88.5	93

(c)

图 8-19 聚集函数实例

对于“每门课程的平均成绩及最高成绩”应该按课程号分组,组内求平均值,并寻找最高成绩。其查询表达式为 $\text{Cno} \text{ } \text{Gavg} \text{ } \text{Grade}$ 和 $\text{max}_{\text{Grade}}(\text{SC})$,结果如图 8-19(c) 所示。

8.2.2 元组演算

元组关系演算是非过程化查询语言。它只描述所需信息,而不给出获得该信息的具体过程。在元组关系演算中,元组关系演算表达式中的变量是以元组为单位的,其一般形式为 $\{t|P(t)\}$ 。其中 t 是元组变量, $P(t)$ 是元组关系演算公式,公式是由原子公式组成的。

1. 原子公式

原子命题函数是公式,简称为原子公式。它有下列 3 种形式:

(1) $R(t)$ 。 R 是关系名, t 是元组变量, $R(t)$ 表示这样一个命题:“ t 是关系 R 的一个元组”。

(2) $t[i]\theta C$ 或 $C\theta t[i]$ 。 $t[i]$ 表示元组变量 t 的第 i 个分量, C 是常量, θ 为算术比较运算符。 $t[i]\theta C$ 或 $C\theta t[i]$ 表示这样一个命题:“元组变量 t 的第 i 个分量与 C 之间满足 θ 运算”。

例如, $t[i]<8$ 表示 t 的第三个分量小于 8。 $t[2]<\text{'数据库'}$ 表示 t 的第二个分量等于“数据库”。

(3) $t[i]\theta u[j]$ 。 t 、 u 是两个元组变量, $t[i]\theta u[j]$ 表示这样一个命题:“元组变量 t 的第 i 个分量与元组变量 u 的第 j 个分量之间满足 θ 运算”。

例如, $t[2]\geq u[4]$ 表示 t 的第二个分量大于等于 u 的第四个分量。

2. 公式的定义

若一个公式中的一个元组变量前有全称量词 \forall 或存在量词 \exists 符号,则称该变量为约束变量,否则称之为自由变量。公式可递归定义如下:

(1) 原子公式是公式。

(2) 如果 φ_1 和 φ_2 是公式,那么, $\neg \varphi_1, \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2$ 和 $\varphi_1 \Rightarrow \varphi_2$ 也都是公式。分别表示如下命题: $\neg \varphi_1$ 表示“ φ_1 不是真”, $\varphi_1 \vee \varphi_2$ 表示“ φ_1 或 φ_2 为真”, $\varphi_1 \wedge \varphi_2$ 表示“ φ_1 和 φ_2 都为真”, $\varphi_1 \Rightarrow \varphi_2$ 表示“若 φ_1 为真则 φ_2 为真”。

(3) 如果 φ_1 是公式,那么, $\exists t(\varphi_1)$ 是公式。 $\exists t(\varphi_1)$ 表示这样一个命题:“若有一个 t 使 φ_1 为真,则 $\exists t(\varphi_1)$ 为真,否则 $\exists t(\varphi_1)$ 为假”。

(4) 如果 φ_1 是公式,那么, $\forall t(\varphi_1)$ 是公式。 $\forall t(\varphi_1)$ 表示这样一个命题:“若所有的 t 使 φ_1 为真,则 $\forall t(\varphi_1)$ 为真,否则 $\forall t(\varphi_1)$ 为假”。

公式中运算符的优先顺序如下:

算术比较运算符 θ 、 \exists 和 \forall 、 \neg 、 \wedge 和 \vee 以及 \Rightarrow 。加括号时,括号中的运算符优先。

3. 把关系代数的五种基本运算转换为元组演算表达式

关系代数表达式可以用元组演算表达式表示。由于任何一个关系代数表达式都可以用 5 种基本的关系运算组合表示,因此,只须给出 5 种基本关系运算的元组演算表达式表示形式即可。

1) 并

并的演算表达式如: $R \cup S = \{t \mid R(t) \vee S(t)\}$

2) 差

差的演算表达式如: $R - S = \{t \mid R(t) \wedge \neg S(t)\}$

3) 笛卡儿积的元组演算表达式

假定关系 R 有 n 个属性,关系 S 有 m 个属性,则 $R \times S$ 后生成的新关系是 $n+m$ 目关系,即有 $n+m$ 个属性。其元组演算表达式为:

$$R \times S = \{t \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge t[1] = u[1] \wedge \cdots \wedge t[n] = u[n] \wedge t[n+1] = v[1] \wedge \cdots \wedge t[n+m] = v[m])\}$$

4) 投影

$$\pi_{i_1, i_2, \dots, i_k}(R) = \{t \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge t[2] = u[i_2] \wedge \cdots \wedge t[k] = u[i_k])\}$$

5) 选择

$$\sigma_F(R) = \{t \mid R(t) \wedge F\}$$

【例 8.10】 设有关系 R 、 S 如图 8-20 所示,对如下的元组演算表达式,求出它们的值。

元组演算表达式如下:

$$R1 = \{t \mid R(t) \wedge \neg S(t)\}$$

$$R2 = \{t \mid (\exists u)(R(t) \wedge S(u) \wedge t[3] < u[2])\}$$

$$R3 = \{t \mid (\forall u)(R(t) \wedge S(u) \wedge t[3] > u[1])\}$$

$$R4 = \{t \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge u[2] > v[1] \wedge t[1] = u[1] \wedge t[2] = v[1] \wedge t[3] = v[3])\}$$

A	B	C	A	B	C
1	2	3	3	7	11
4	5	6	4	5	6
7	8	9	5	9	13
10	11	12	6	10	14

R

S

图 8-20 关系 R、S

解：R1 = {t | R(t) ∧ ¬ S(t)} 元组演算表达式的含义为：新生成的关系 R1 中的元组来自关系 R，但该元组又不在关系 S 中。查询结果如图 8-21(a) 所示。

R2 = {t | (∃ u)(R(t) ∧ S(u) ∧ t[3] < u[2])} 元组演算表达式的含义为：新生成的关系 R2 中的元组来自关系 R，但该元组的第三个分量值必须小于关系 S 中某个元组的第二个分量值。查询结果如图 8-21(b) 所示。

R3 = {t | (∀ u)(R(t) ∧ S(u) ∧ t[3] > u[1])} 元组演算表达式的含义为：新生成的关系 R3 中的元组来自关系 R，但该元组的第三个分量值必须小于关系 S 中任意一个元组的第一个分量值。查询结果如图 8-21(c) 所示。

R4 = {t | (∃ u)(∃ v)(R(u) ∧ S(v) ∧ u[2] > v[1] ∧ t[1] = u[1] ∧ t[2] = v[1] ∧ t[3] = v[3])} 元组演算表达式的含义为：新生成的关系 R4 中的元组来自关系 R 和 S，因为结果集元组变量的三个分量 t[1]、t[2] 和 t[3] 应满足 t[1] = u[1]、t[2] = v[1] 和 t[3] = v[3]，分别表示第一个分量为关系 R 的第一个分量，第二个分量为关系 S 的第一个分量，第三个分量为关系 S 的第三个分量。查询结果如图 8-21(d) 所示。

【例 8.11】 设学生课程数据库中有 3 个关系(学生关系 S、课程关系 C 以及学生选课关系 SC)如图 8-9 所示。请用元组演算表达式完成例 8.6 中的查询问题。

解：

(1) 检索选修课程名为“数学”的学生号和学生姓名的元组演算表达式为：

$$\{t \mid (\exists u)(\exists v)(\exists w)(S(u) \wedge SC(v) \wedge C(w) \wedge u[1] = v[1] \wedge v[2] = w[1] \wedge w[2] = \text{'数学'} \wedge t[1] = u[1] \wedge t[2] = u[2])\}$$

(2) 检索至少选修了课程号为‘1’和‘3’的学生号的元组演算表达式为：

$$\{t \mid (\exists u)(\exists v)(SC(u) \wedge SC(v) \wedge u[1] = v[1] \wedge u[2] = \text{'1'} \wedge v[2] = \text{'3'} \wedge t[1] = u[1])\}$$

A	B	C
1	2	3
7	8	9
10	11	12

(a) R1

A	B	C
1	2	3
4	5	6
7	8	9

(b) R2

A	B	C
7	8	9
10	11	12

(c) R3

R.A	S.A	S.C
4	3	11
4	4	6
7	3	11
7	4	6
7	5	13
7	6	14
10	3	11
10	4	6
10	5	13
10	6	14

(d) R4

图 8-21 R1、R2、R3 和 R4

(3) 检索选修了‘操作系统’或‘数据库’课程的学号和姓名的元组演算表达式为:

$$\begin{aligned} & \{t \mid (\exists u)(\exists v)(\exists w)(S(u) \wedge SC(v) \wedge C(w) \wedge u[1] = v[1] \wedge v[2] \\ & \quad = w[1] \wedge (w[2] = \text{'操作系统'} \vee w[2] = \text{'数据库'}) \wedge t[1] \\ & \quad = u[1] \wedge t[2] = u[2])\} \end{aligned}$$

(4) 检索年龄在 18 到 20 之间(含 18 和 20)的女生的学号、姓名及年龄的元组演算表达式为:

$$\begin{aligned} & \{t \mid (\exists u)(S(u) \wedge u[5] \geq \text{'18'} \wedge u[5] \leq \text{'20'} \wedge u[3] = \text{'女'} \wedge t[1] \\ & \quad = u[1] \wedge t[2] = u[2] \wedge t[3] = u[5])\} \end{aligned}$$

(5) 检索选修了“数据库”课程的学生学号、姓名及成绩的元组演算表达式为:

$$\begin{aligned} & \{t \mid (\exists u)(\exists v)(\exists w)(S(u) \wedge SC(v) \wedge C(w) \wedge u[1] = v[1] \wedge v[2] \\ & \quad = w[1] \wedge w[2] = \text{'数据库'} \wedge t[1] = u[1] \wedge t[2] \\ & \quad = u[2] \wedge t[3] = v[3])\} \end{aligned}$$

(6) 检索选修所有课程的学生姓名及所在系的元组演算表达式为:

$$\begin{aligned} & \{t \mid (\exists u)(\forall v)(\exists w)(S(u) \wedge C(v) \wedge SC(w) \wedge u[1] = w[1] \wedge w[2] \\ & \quad = v[1] \wedge t[1] = u[2] \wedge t[2] = u[4])\} \end{aligned}$$

(7) 检索选修课程包括“1042”号学生所学课程的学生学号的元组演算表达式为:

$$\begin{aligned} & \{t \mid (\exists v)(SC(u) \wedge (\forall v)(SC(v) \wedge (v[1] = \text{'1042'} \Rightarrow (\exists w)(SC(w) \wedge w[1] \\ & \quad = u[1] \wedge w[2] = v[2]))) \wedge t[1] = u[1])\} \end{aligned}$$

这里的解题思路是,在 SC 关系中依次检查“1042”号所选修的课程,再看某一个学生

是否也选修了该门课。如果对于“1042”所选修的每门课程该学生都选修了,则该学生为满足条件的学生,将所有的学生都找出来即完成了本题的要求。

(8) 检索不选修 2 号课程的学生姓名和所在系的元组演算表达式为:

$$\{t \mid (\exists u)(\forall v)(S(u) \wedge SC(v) \wedge (u[1] = v[1] \Rightarrow \wedge v[2] \neq '2') \wedge t[1] = u[1] \wedge t[2] = u[4])\}$$

注意:当查询涉及到否定或所有值时,就要用到差运算或除法运算。

8.2.3 域演算

域关系演算简称域演算。在域演算中,表达式中的变量是表示域的变量。可将关系的属性名视为域变量。

域演算表达式的一般形式为:

$$\{t_1, \dots, t_k \mid P(t_1, \dots, t_k)\}$$

其中, t_1, \dots, t_k 是域变量, $P(t_1, \dots, t_k)$ 是域演算公式。

1. 原子公式

原子命题函数是公式,简称为原子公式。它有下列 3 种形式。

(1) $R(t_1, \dots, t_i, \dots, t_k)$ 。R 是 k 元关系, t_i 是元组变量 t 的第 i 个分量, $R(t_1, \dots, t_i, \dots, t_k)$ 表示这样一个命题:“以 $t_1, \dots, t_i, \dots, t_k$ 为分量的元组在关系 R 中”。

(2) $t_i \theta C$ 或 $C \theta t_i$ 。 t_i 表示元组变量 t 的第 i 个分量, C 是常量, θ 为算术比较运算符。

(3) $t_i \theta u_j$ 。 t_i, u_j 是两个域变量, t_i 是元组变量 t 的第 i 个分量, u_j 是元组变量 u 的第 j 个分量,它们之间应满足 θ 运算。

例如, $t_1 \geq u_4$ 表示 t 的第一个分量值大于等于 u 的第四个分量值。

2. 公式的定义

若公式中的一个元组变量前有全称量词 \forall 或存在量词 \exists 符号,则称该变量为约束变量,否则称之为自由变量。公式可递归定义如下:

(1) 原子公式是公式。

(2) 如果 φ_1 和 φ_2 是公式,那么, $\neg \varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ 和 $\varphi_1 \Rightarrow \varphi_2$ 也都是公式。

(3) 如果 φ_1 是公式,那么, $\exists t_i(\varphi_1)$ 是公式。 $\exists t_i(\varphi_1)$ 表示这样一个命题:“若有一个 t_i 使 φ_1 为真,则 $\exists t_i(\varphi_1)$ 为真,否则 $\exists t_i(\varphi_1)$ 为假”。

(4) 如果 $\varphi_1(t_1, \dots, t_i, \dots, t_k)$ 是公式,那么, $\forall t_i(\varphi_1)$ 是公式。 $\forall t_i(\varphi_1)$ 表示这样一个命题:“若所有的 t_i 使 $\varphi_1(t_1, \dots, t_i, \dots, t_k)$ 为真,则 $\forall t_i(\varphi_1)$ 为真,否则 $\forall t_i(\varphi_1)$ 为假”。

公式中运算符的优先顺序如下:

算术比较运算符 θ 、 \exists 和 \forall 、 \neg 、 \wedge 和 \vee 以及 \Rightarrow 。加括号时,括号中的运算符优先。

【例 8.12】 设有关系 R、S 如图 8-22 所示,对下列域演算表达式,求出它们的值。

A	B	C
1	2	1
4	5	7
7	8	6
10	11	9

R

A	B	C
1	2	3
4	5	6
7	8	9
10	11	11

S

图 8-22 关系 R 与 S

域演算表达式:

$$R1 = \{t_1 t_2 t_3 \mid R(t_1 t_2 t_3) \wedge t_1 < t_2 \wedge t_2 > t_3\}$$

$$R2 = \{t_1 t_2 t_3 \mid (R(t_1 t_2 t_3) \wedge t_1 > 4) \vee (S(t_1 t_2 t_3) \wedge t_2 < 8)\}$$

$$R3 = \{t_1 t_2 t_3 \mid (\exists u)(\exists v)(\exists w)R(u_2 v) \wedge S(t_1 w t_3) \wedge u > 7 \wedge r > w\}$$

解: R1、R2 和 R3 的运算结果见图 8-23。

A	B	C
1	2	1
7	8	6
10	11	9

(a) R1

A	B	C
7	8	6
10	11	9
1	2	3
4	5	6

(b) R2

S.A	R.B	S.C
1	8	3
4	8	6
1	11	3
4	11	6
7	11	9

(c) R3

图 8-23 关系 R1、R2 与 R3

【例 8.13】 图 8-9 学生数据库中有 S、SC 和 C 3 个关系, 请用域演算表达式完成如下查询:

- (1) 检索年龄小于 20 岁的男生的学号和姓名。
- (2) 检索选修了‘数据库’课程的学号和姓名。

解:

- (1) 检索年龄小于 20 岁的男生的学号和姓名的域演算表达式为:

$$\{t_1 t_2 \mid (\exists u_1)(\exists u_2)(\exists u_3)(\exists u_4)(\exists u_5)(S(u_1 u_2 u_3 u_4 u_5) \wedge u_3 = \text{'男'} \wedge u_5 < \text{'20'} \wedge t_1 = u_1 \wedge t_2 = u_2)\}$$

注意: 为了书写方便, $(\exists u_1)(\exists u_2)(\exists u_3)(\exists u_4)(\exists u_5)$ 可用 $(\exists u_1 u_2 u_3 u_4 u_5)$ 的简化表示形式。

另外, 在上述表达式中由于 S 关系中的性别是一个常量‘男’, 因此也可以将表达式简化为 $\{t_1 t_2 \mid (\exists u_4)(\exists u_5)(S(t_1 t_2 \text{'男'} u_4 u_5) \wedge u_5 < \text{'20'})\}$ 。

- (2) 检索选修了‘数据库’课程的学号和姓名的域演算表达式为:

$$\{t_1 t_2 \mid (\exists u_1 u_2 u_3 u_4 u_5)(\exists v_1 v_2 v_3)(\exists w_1 w_2 w_3 w_4)(S(u_1 u_2 u_3 u_4 u_5) \wedge SC(v_1 v_2 v_3) \wedge C(w_1 w_2 w_3 w_4) \wedge u_1 = v_1 \wedge v_2 = w_1 \wedge w_2 = '数据库' \wedge t_1 = u_1 \wedge t_2 = u_2)\}$$

上述表达式可以简化为：

$$\{t_1 t_2 \mid (\exists u_3 u_4 u_5)(\exists v_2 v_3)(\exists w_1 w_3 w_4)(S(t_1 t_2 u_3 u_4 u_5) \wedge SC(t_1 v_2 v_3) \wedge C(w_1 '数据库' w_3 w_4) \wedge v_2 = w_1)\}$$

8.3 查询优化

8.3.1 基本概念

- 查询处理：是指从数据库中提取数据的一系列活动。这一系列活动包括：将高级数据库语言表示的查询语句翻译成为能在文件系统这一物理层次上实现的表达式，为优化查询进行各种转换，以及查询的实际执行。
- 查询处理的代价：通常取决于磁盘的访问，因为磁盘比内存的访问速度要慢。对于一个给定的查询，可以有許多可能的处理策略，复杂查询更是如此。就所需的磁盘访问次数而言，策略好坏差别很大，有时甚至相差几个数量级。所以，系统多花一点时间选择一个较好的查询策略是很值得的。
- 查询优化：是为了查询选择最有效的查询计划的过程。查询优化一方面在关系代数级进行优化，力图找出与给定表达式等价，但执行效率更高的一个表达式。查询优化的另一方面涉及查询语句处理策略的选择，例如选择采用的具体算法以及使用的特定索引等。

一个查询往往会有许多实现办法，关键是如何找出一个与之等价的且操作时间又少的表达式。下面将专门讨论这个问题。

8.3.2 关系代数表达式中的查询优化

关系系统的查询优化是关系数据库管理系统实现的关键技术，又是关系系统的优点。因为，用户只要提出“干什么”，不必指出“怎么干”。

在关系代数表达式中，需要指出若干关系的操作步骤。问题是怎样做才能保证省时、省空间以及效率高，这就是查询优化的问题。

需要注意的是，在关系代数运算中，笛卡儿积、连接运算最费时间和空间，究竟应采用什么样的策略，能够节省时间空间，这就是优化的准则。

1. 优化的准则

- 提早执行选取运算。对于有选择运算的表达式，优化的原则是尽可能先执行选择

运算的等价表达式,以得到较小的中间结果,减少运算量和从外存读块的次数。

- 合并乘积与其后的选择运算为连接运算。在表达式中,当乘积运算后面是选择运算时,应该合并为连接运算,使选择与乘积一道完成,以避免做完乘积后,再对一个大的乘积关系进行选择运算。
- 将投影运算与其后的其他运算同时进行,以避免重复扫描关系。
- 将投影运算和其前后的二目运算结合起来,使得没有必要为去掉某些字段再扫描一遍关系。
- 在执行连接前对关系做适当的预处理,就能快速找到要连接的元组。方法有两种:即索引连接法和排序合并连接法。
- 存储公共子表达式。公共子表达式的结果应存于外存(中间结果),这样,当从外存读出它的时间比计算时间少时,就可节约操作时间。

2. 关系代数表达式的等价变换规则

优化的策略均涉及关系代数表达式,所以讨论关系代数表达式的等价变换规则显得十分重要。常用的等价变换规则有如下 10 种。

1) 连接、笛卡儿积交换率

设 E_1 和 E_2 是关系代数表达式, F 是连接运算的条件,则有:

$$E_1 \times E_2 \equiv E_2 \times E_1$$

$$E_1 \bowtie_F E_2 \equiv E_2 \bowtie_F E_1$$

2) 连接、笛卡儿积结合率

设 E_1, E_2 和 E_3 是关系代数表达式, F_1 和 F_2 是连接运算的条件,则有:

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie_{F_1} E_2) \bowtie_{F_2} E_3 \equiv E_1 \bowtie_{F_1} (E_2 \bowtie_{F_2} E_3)$$

3) 投影的串接定律

设 E 是关系代数表达式, A_1, A_2, \dots, A_n 和 B_1, B_2, \dots, B_m 是属性名,且 B_1, B_2, \dots, B_m 是 A_1, A_2, \dots, A_n 的子集。则有:

$$\prod_{A_1, A_2, \dots, A_n} (\prod_{B_1, B_2, \dots, B_m} (E)) \equiv \prod_{A_1, A_2, \dots, A_n} (E)$$

该规则的目的是使一些投影消失。

4) 选择的串接定律

设 E 是关系代数表达式, F_1 和 F_2 是选取条件表达式,选择的串接定律说明选择条件可以合并,于是有:

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

5) 选择与投影的交换律

设 E 是关系代数表达式, F 是选取条件表达式, 并且只涉及 A_1, A_2, \dots, A_n 属性, 则有:

$$\sigma_F(\prod_{A_1, A_2, \dots, A_n}(E)) \equiv \prod_{A_1, A_2, \dots, A_n}(\sigma_F(E))$$

若 F 中有不属于 A_1, A_2, \dots, A_n 的属性 (B_1, B_2, \dots, B_m) , 那么有更一般的规则:

$$\sigma_F(\prod_{A_1, A_2, \dots, A_n}(E)) \equiv \prod_{A_1, A_2, \dots, A_n}(\sigma_F(\prod_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E)))$$

该规则可将投影分裂为两个, 使得其中的一个可能被移到树的叶端。

6) 选择与笛卡儿积的交换律

若 F 涉及的都是 E_1 中的属性, 则:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

如果 $F = F_1 \wedge F_2$, 并且 F_1 只涉及 E_1 中的属性, F_2 只涉及 E_2 中的属性, 则有:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

7) 选择与并的交换律

设 $E = E_1 \cup E_2$, E_1 和 E_2 有相同的属性, 则:

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

8) 选择与差的交换律

设 E_1 和 E_2 有相同的属性, 则:

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

9) 投影与笛卡儿积的交换律

设 E_1 和 E_2 是两个关系表达式, A_1, A_2, \dots, A_n 是 E_1 中的属性, B_1, B_2, \dots, B_m 是 E_2 中的属性, 则:

$$\prod_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E_1 \times E_2) \equiv \prod_{A_1, A_2, \dots, A_n}(E_1) \times \prod_{B_1, B_2, \dots, B_m}(E_2)$$

10) 投影与并的交换律

设 E_1 和 E_2 有相同的属性, 则:

$$\prod_{A_1, A_2, \dots, A_n}(E_1 \cup E_2) \equiv \prod_{A_1, A_2, \dots, A_n}(E_1) \cup \prod_{A_1, A_2, \dots, A_n}(E_2)$$

3. 查询优化的算法

算法: 关系代数表达式的优化。

输入: 一个关系代数表达式的语法树。

输出: 计算该表达式的程序。

方法:

(1) 利用规则 4 将形如 $\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_n}(E)$ 的表达式变换为

$$\sigma_{F_1}(\sigma_{F_2}(\dots(\sigma_{F_n}(E))\dots))$$

(2) 对每一个选择, 利用规则 4~8 尽可能将它移到树的叶端。

- (3) 对每一个投影,利用规则 3、9、10 和 5 中的一般形式尽可能将它移到树的叶端。
- (4) 利用规则 3~5 将选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。使多个选择或投影能同时进行,或在一次扫描中全部完成。
- (5) 将上述得到的语法树的内节点分组。每一双目运算(\times , \cup , \bowtie 和 $-$)和它所有的直接祖先为一组(这些直接祖先是 σ 和 π 运算)。如果其后代直到叶子全部是单目运算,则将它并入该组。
- (6) 生成一个程序,每组节点的计算是程序中的一步。各步的顺序是任意的,只要保证任何一组的计算不会在它的后代组之前计算即可。

【例 8.14】 供应商数据库中有供应商、零件、项目和供应 4 个基本表(关系):

S(Sno, Sname, Status, City);
P(Pno, Pname, Color, Weight);
J(Jno, Jname, City);
SPJ(Sno, Pno, Jno, Qty).

用户有一查询语句:检索使用上海供应商生产的红色零件的工程号。

- (1) 试写出该查询的关系代数表达式;
- (2) 试写出查询优化的关系代数表达式;
- (3) 画出该查询初始的关系代数表达式的语法树;
- (4) 使用优化算法,对语法树进行优化,并画出优化后的语法树。

解:

- (1) 试写出该查询的关系代数表达式。

$$\pi_{Jno}(\sigma_{City='上海' \wedge Color='红'}(S \bowtie SPJ \bowtie P))$$

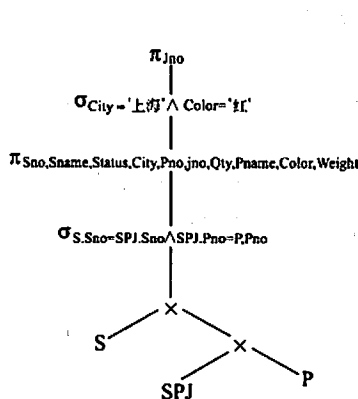


图 8-24 优化前

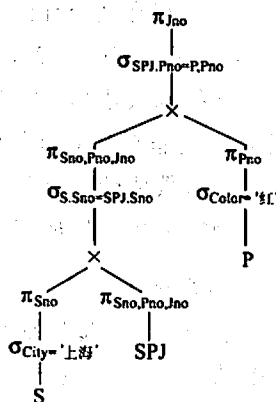


图 8-25 优化后

(2) 试写出查询优化的关系代数表达式。

$\pi_{jno}(\pi_{sno}(\sigma_{City='上海'}(S)) \bowtie \pi_{sno,jno}(\sigma_{Color='红'}(P)))$

(3) 画出该查询初始的关系代数表达式的语法树(见图 8-24)。

(4) 使用优化算法,对语法树进行优化,并画出优化后的语法树(见图 8-25)。

8.4 关系数据库设计的基础理论

在关系模型中,一个数据库模式是关系模式的集合。关系数据理论是指导数据库设计的基础,关系数据库设计是数据库语义学的问题。要保证构造的关系既能准确地反映现实世界,又有利于应用和具体的操作。关系数据库设计理论的核心是数据间的函数依赖,衡量的标准是关系规范化的程度及分解的无损连接和保持函数依赖性。关系数据库设计的目标是生成一组合适的、性能良好的关系模式,以减少系统中信息存储的冗余度,但又可方便地获取信息。

8.4.1 基础知识

1. 函数的依赖关系

数据依赖是通过一个关系中属性之间值的相等与否体现出来的数据间的相互关系,是现实世界属性间的联系和约束的抽象,是数据内在的性质,是语义的体现。函数依赖则是一种最重要、最基本的数据依赖。

1) 函数依赖

设 $R(U)$ 是属性集 U 上的关系模式, X 和 Y 是 U 的子集。若 $R(U)$ 的任何一个可能的关系 r 中不可能存在两个元组在 X 上的属性值相等,而在 Y 上的属性值不等,则称 X 函数决定 Y 或 Y 函数依赖于 X , 记作: $X \rightarrow Y$ 。

(1) 非平凡的函数依赖: 如果 $X \rightarrow Y$, 但 $Y \not\subseteq X$, 则称 $X \rightarrow Y$ 是非平凡的函数依赖。一般情况下总是讨论非平凡的函数依赖。

(2) 平凡的函数依赖: 如果 $X \rightarrow Y$, 但 $Y \subseteq X$, 则称 $X \rightarrow Y$ 是平凡的函数依赖。

(3) 完全函数依赖: 在 $R(U)$ 中, 如果 $X \rightarrow Y$, 并且对于 X 的任何一个真子集 X' , 都有 $X' \not\rightarrow Y$, 不能决定 Y , 则称 Y 对 X 完全函数依赖, 记作: $X \xrightarrow{f} Y$ 。

例如, 给定一个学生选课关系 $SC(Sno, Cno, G)$, 可以得到 $F = \{(Sno, Cno) \rightarrow G\}$, 对 (Sno, Cno) 中的任何一个真子集 Sno 或 Cno 都不能决定 G , 所以, G 完全依赖于 Sno 和 Cno 。

(4) 部分函数依赖: 如果 $X \rightarrow Y$, 但 Y 不完全函数依赖于 X , 则称 Y 对 X 部分函数依

赖,记作: $X \xrightarrow{P} Y$ 。部分函数依赖也称局部函数依赖。

(5) 传递依赖: 在 $R(U, F)$ 中, 如果 $X \rightarrow Y, Y \not\subseteq X, Y \leftrightarrow X$ 以及 $Y \rightarrow Z$, 则称 Z 对 X 传递依赖。

- 码: 设 K 为 $R(U, F)$ 中属性的组合, 若 $K \rightarrow U$, 且对于 K 的任何一个真子集 K' , 都有 K' 不能决定 U , 则 K 为 R 的候选码。若有多个候选码, 则选一个作为主码。候选码通常也称候选关键字。
- 主属性和非主属性: 包含在任何一个候选码中的属性叫做主属性, 否则叫做非主属性。
- 外码: 若 $R(U)$ 中的属性或属性组 X 非 R 的码, 但 X 是另一个关系的码, 则称 X 为外码。
- 多值依赖定义: 若关系模式 $R(U)$ 中, X, Y, Z 是 U 的子集, 并且 $Z = U - X - Y$ 。当且仅当对 $R(U)$ 的任何一个关系 r , 给定一对 (x, z) 值, 有一组 Y 的值, 这组值仅仅决定于 x 值而与 z 值无关, 则称“ Y 多值依赖于 X ”或“ X 多值决定 Y ”成立。记为 $X \twoheadrightarrow Y$ 。

多值依赖具有如下 6 种性质:

- (1) 多值依赖具有对称性。若 $X \twoheadrightarrow Y$, 则 $X \twoheadrightarrow Z$, 其中 $Z = U - X - Y$ 。
- (2) 多值依赖的传递性。若 $X \twoheadrightarrow Y, Y \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Z - Y$ 。
- (3) 函数依赖可以看作多值依赖的特殊情况。
- (4) 若 $X \twoheadrightarrow Y, X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow YZ$ 。
- (5) 若 $X \twoheadrightarrow Y, X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Y \cap Z$ 。
- (6) 若 $X \twoheadrightarrow Y, X \twoheadrightarrow Z$, 则 $X \twoheadrightarrow Z - Y$ 。

2) 函数依赖的公理系统 (armstrong 公理系统)

设关系模式 $R(U, F)$, 其中 U 为属性集, F 是 U 上的一组函数依赖, 那么有如下推理规则。

- (1) 自反律: 若 $Y \subseteq X \subseteq U$, 则 $X \rightarrow Y$ 为 F 所蕴涵。
- (2) 增广律: 若 $X \rightarrow Y$ 为 F 所蕴涵, 且 $Z \subseteq U$, 则 $XZ \rightarrow YZ$ 为 F 所蕴涵。
- (3) 传递律: 若 $X \rightarrow Y, Y \rightarrow Z$ 为 F 所蕴涵, 则 $X \rightarrow Z$ 为 F 所蕴涵。

根据上述 3 条推理规则又可推出下述 3 条推理规则。

- (1) 合并规则: 若 $X \rightarrow Y, X \rightarrow Z$, 则 $X \rightarrow YZ$ 为 F 所蕴涵。
- (2) 伪传递率: 若 $X \rightarrow Y, WY \rightarrow Z$, 则 $XW \rightarrow Z$ 为 F 所蕴涵。
- (3) 分解规则: 若 $X \rightarrow Y, Z \subseteq Y$, 则 $X \rightarrow Z$ 为 F 所蕴涵。

引理: $X \rightarrow A_1 A_2 \cdots A_k$ 成立的充分必要条件是 $X \rightarrow A_i$ 成立 ($i = 1, 2, \cdots, k$)。证明略。

2. 函数依赖的闭包 F^+ 及属性的闭包 X_F^+

1) 函数依赖的闭包

定义：关系模式 $R(U, F)$ 中为 F 所逻辑蕴涵的函数依赖的全体称为 F 的闭包，记为 F^+ 。

2) 属性的闭包 X_F^+

定义：设 F 为属性集 U 上的一组函数依赖， $X \subseteq U$ ， $X_F^+ = \{A \mid X \rightarrow A \text{ 能由 } F \text{ 根据 Armstrong 公理导出}\}$ ，则称 X_F^+ 为属性集 X 关于函数依赖集 F 的闭包。

算法：求属性的闭包 X_F^+ 。

输入： X, F

输出： X_F^+

步骤：

(1) 令 $X^{(0)} = X, I = 0$;

(2) 求 $B, B = \{A \mid (\exists v)(\exists w)(V \rightarrow W \in F \wedge V \subseteq X^{(i)} \wedge A \in W)\}$;

(3) $X^{(i+1)} = B \cup X^{(i)}$;

(4) 判断 $X^{(i+1)} = X^{(i)}$ 是否成立;

(5) 若相等，或 $X^{(i)} = U$ ，则 $X^{(i)}$ 为属性集 X 关于函数依赖集 F 的闭包，且算法终止;

(6) 若不相等，则 $i = i + 1$ ，返回第二步。

【例 8.15】已知关系模式 $R(U, F)$ ， $U = \{A, B, C, D, E\}$ ， $F = \{A \rightarrow B, D \rightarrow C, BC \rightarrow E, AC \rightarrow B\}$ 。求 $(AE)_F^+$ 和 $(AD)_F^+$ 。

解：求 $(AE)_F^+$ ，根据上述算法，设 $X^{(0)} = AE$ 。

计算 $X^{(1)}$ ：逐一扫描 F 中的各个函数依赖，找到左部为 A, E 或 AE 的函数依赖，找到一个 $A \rightarrow B$ 。故有 $X^{(1)} = AE \cup B$ 。

计算 $X^{(2)}$ ：逐一扫描 F 中的各个函数依赖，找到左部为 ABE 或 ABE 子集的函数依赖，因为找不到这样的函数依赖，所以， $X^{(1)} = X^{(2)}$ 。算法终止， $(AE)_F^+ = ABE$ 。

求 $(AD)_F^+$ ，根据上述算法，设 $X^{(0)} = AD$ 。

计算 $X^{(1)}$ ：逐一扫描 F 中的各个函数依赖，找到左部为 A, D 或 AD 的函数依赖，找到 $A \rightarrow B$ 和 $D \rightarrow C$ 两个函数依赖。故有 $X^{(1)} = AD \cup BC$ 。

计算 $X^{(2)}$ ：逐一扫描 F 中的各个函数依赖，找到左部为 $ADBC$ 或 $ADBC$ 子集的函数依赖，得到 $BC \rightarrow E$ 和 $AC \rightarrow B$ 两个函数依赖。故有 $X^{(2)} = ABCD \cup E$ 。所以， $X^{(2)} = ABCDE = U$ 。算法终止， $(AD)_F^+ = ABCDE$ 。

3. 候选码的求解方法

给定一个关系模式 $R(U, F)$ ， $U = \{A_1, A_2, \dots, A_n\}$ ， F 是 R 的函数依赖集，则属性可以分为如下 4 类。

- L : 仅出现在函数依赖集 F 左部的属性。
- R : 仅出现在函数依赖集 F 右部的属性。
- LR : 在函数依赖集 F 左右都出现的属性。
- NLR : 在函数依赖集 F 左右都未出现的属性。

根据候选码的特性可以得出如下结论:

(1) 给定一个关系模式 $R(U, F)$, 若 $X(X \subseteq U)$ 是 L 类属性, 则 X 必为 R 的任一候选码的成员。若 $X_F^+ = U$, 则 X 必为 R 的惟一候选码。

(2) 给定一个关系模式 $R(U, F)$, 若 $X(X \subseteq U)$ 是 R 类属性, 则 X 不是 R 的任一候选码的成员。

(3) 给定一个关系模式 $R(U, F)$, 若 $X(X \subseteq U)$ 是 NLR 类属性, 则 X 必为 R 的任一候选码的成员。

(4) 给定一个关系模式 $R(U, F)$, 若 $X(X \subseteq U)$ 是 L 类和 NLR 类属性组成的属性集, 且 $X_F^+ = U$, 则 X 必为 R 的惟一候选码。

【例 8.16】 设有关系模式 $R(U, F)$, 其中 $U = (A, B, C, D)$, $F = \{A \rightarrow C, C \rightarrow B, AD \rightarrow B\}$ 。求 R 的候选码。

解: 根据结论(1)可以求得 R 的候选码为 AD , 而且 AD 是 R 惟一的候选码。分析如下:

(1) 检查 F 发现, A 和 D 只出现在函数依赖的左部, 所以为 L 类属性, 而 F 包含全属性, 即不存在 NLR 类的属性;

(2) 根据求属性闭包的算法, 由 F 中 $A \rightarrow C$ 和 $AD \rightarrow B$ 可以求得 $(AD)_F^+ = ABCD = U$ 。而 AD 中不存在一个真子集能决定全属性, 故 AD 为 R 的候选码。

【例 8.17】 设有关系模式 $R(U, F)$, 其中 $U = (H, I, J, K, L, M)$, $F = \{H \rightarrow I, K \rightarrow I, LM \rightarrow K, I \rightarrow K, KH \rightarrow M\}$ 。求 R 的候选码。

解: 根据结论(1)、(2)、(3)和(4)可以求得 R 的候选码为 HLJ , 而且 HLJ 是 R 惟一的候选码。现分析如下:

(1) 检查 F 发现, H 和 L 只出现在函数依赖的左部, 所以为 L 类属性。 J 为 NLR 类的属性。

(2) 根据求属性闭包的算法, 由 F 中 $H \rightarrow I, I \rightarrow K$ 和 $KH \rightarrow M$ 可以求得 $(HLJ)_F^+ = HIJKLM = U$ 。而 HLJ 中不存在一个真子集能决定全属性, 故 HLJ 为 R 的候选码。

8.4.2 规范化

关系数据库的设计方法之一就是设计满足适当范式的模式, 通常可以通过判断分解后的模式达到几范式来评价模式规范化的程度。范式有: $1NF$ 、 $2NF$ 、 $3NF$ 、 $BCNF$ 、 $4NF$ 和

5NF,其中 1NF 级别最低。这几种范式之间 $1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$ 成立。

通过分解,可以将一个低一级范式的关系模式转换成若干个高一级范式的关系模式,这种过程叫做规范化。下面将给出各个范式的定义。

1. 1NF(第一范式)

定义:若关系模式 R 的每一个分量是不可再分的数据项,则关系模式 R 属于第一范式(1NF)。记为 $R \in 1NF$ 。

例如,对于供应者和它所提供的零件信息,其关系模式 FIRST 和函数依赖集 F 如下:

FIRST(Sno, Sname, Status, City, Pno, Qty)

$F = \{ Sno \rightarrow Sname, Sno \rightarrow Status, Status \rightarrow City, (Sno, Pno) \rightarrow Qty \}$

具体的关系 FIRST 如表 8-2 所示。从表 8-2 中可以看出,每一个分量都是不可再分的数据项,所以是 1NF 的。但是,1NF 存在下列 4 个问题。

表 8-2 FIRST

Sno	Sname	Status	City	Pno	Qty
S1	精益	20	天津	P1	200
S1	精益	20	天津	P2	300
S1	精益	20	天津	P3	480
S2	盛锡	10	北京	P2	168
S2	盛锡	10	北京	P3	500
S3	东方红	30	北京	P1	300
S3	东方红	30	北京	P2	280
S4	泰达	40	上海	P2	460

(1) 冗余度大:例如,每个供应者的 Sno, Sname, Status 和 City 属性与其供应的零件的种类一样多。

(2) 修改操作引起数据的不一致性:例如,假设供应者 S1 从“天津”搬到“上海”,若稍不注意,就会使一些数据被修改,另一些数据没有被修改,导致数据修改的不一致性。

(3) 插入异常:关系模式 FIRST 的主码为 Sno 和 Pno。按照关系模式实体完整性的规定,主码不能取空值或部分取空值。因此,当供应者的某些信息未提供时(如 Pno),则不能执行插入操作,这就是所谓的插入异常。

(4) 删除异常:若供应商 S4 的 P2 零件销售完了,并且以后不再销售 P2 零件,那么应删除该元组。这样一来,基本关系 FIRST 中就找不到 S4 了,可 S4 又是客观存在的。

正因为上述 4 个原因,所以要对模式进行分解,并引入 2NF。

2. 2NF(第二范式)

定义:若关系模式 $R \in 1NF$,且每一个非主属性完全依赖于码,则关系模式 $R \in 2NF$ 。

换句话说,当 1NF 消除了非主属性对码的部分函数依赖,则称为 2NF。

例如, FIRST 关系中的码是 Sno 和 Pno,而 $Sno \rightarrow Status$,因此非主属性 Status 部分函数依赖于码,故非 2NF 的。

若此时将 FIRST 关系分解为:

$FIRST_1(Sno, Sname, Status, City) \in 2NF$

$FIRST_2(Sno, Pno, Qty) \in 2NF$

则因为分解后的关系模式 $FIRST_1$ 的码为 Sno,非主属性 Sname、Status 和 City 完全依赖于码 Sno,所以属于 2NF。关系模式 $FIRST_2$ 的码为 Sno 和 Pno,非主属性 Qty 完全依赖于码,所以也属于 2NF。

3. 3NF(第三范式)

定义:在关系模式 $R(U, F)$ 中,若不存在这样的码 X ,属性组 Y 及非主属性 $Z (Z \not\subseteq Y)$,使得 $X \rightarrow Y$ 和 $(Y \rightarrow X)Y \rightarrow Z$ 成立,则关系模式 $R \in 3NF$ 。

即当 2NF 消除了非主属性对码的传递函数依赖,则称为 3NF。

例如, $FIRST_1 \notin 3NF$,因为在分解后的关系模式 $FIRST_1$ 中有 $Sno \rightarrow Status$ 和 $Status \rightarrow City$,存在着非主属性 City 传递依赖于码 Sno。若此时将 $FIRST_1$ 继续分解为:

$FIRST_{11}(Sno, Sname, Status) \in 3NF$

$FIRST_{12}(Status, City) \in 3NF$

则通过上述分解,数据库模式 FIRST 转换为 $FIRST_{11}(Sno; Sname, Status)$, $FIRST_{12}(Status, City)$ 和 $FIRST_2(Sno, Pno, Qty)$ 3 个子模式。由于这 3 个子模式都达到了 3NF,因此称分解后的数据库模式达到了 3NF。

可以证明,3NF 模式必是 2NF 模式。产生冗余和异常的两个重要原因是部分依赖和传递依赖。因为 3NF 模式中不存在非主属性对码的部分函数依赖和传递函数依赖,所以具有较好的性能。非 3NF 的 1NF 和 2NF 的性能较弱,一般不宜作为数据库模式。通常要将它们变换为 3NF 或更高级别的范式,这种变换过程称为“关系模式的规范化处理”。

4. BCNF(巴克斯范式)

定义:若关系模式 $R \in 1NF$,且 $X \rightarrow Y$ 和 $Y \not\subseteq X$ 时, X 必含有码,则关系模式 $R \in BCNF$ 。

即当 3NF 消除了主属性对码的部分和传递函数依赖,则称为 BCNF。

结论:一个满足 BCNF 的关系模式应有如下性质:

- (1) 所有非主属性对每一个码都是完全函数依赖;
- (2) 所有非主属性对每一个不包含它的码,也是完全函数依赖;
- (3) 没有任何属性完全函数依赖于非码的任何一组属性。

例如,设 $R(Pno, Pname, Mname)$ 的属性分别表示零件号、零件名和厂商名。如果约

定每种零件号只有一个零件名,但不同的零件号可以有相同的零件名,每种零件可以有多个厂商生产,但每家厂商生产的零件应有不同的零件名。这样可以得到下列一组函数依赖:

$Pno \rightarrow Pname, (Pname, Mname) \rightarrow Pno$

由于该关系模式 R 中的候选码为 $(Pname, Mname)$ 或 $(Pno, Mname)$, 因而关系模式 R 的属性都是主属性, 不存在非主属性对码的传递依赖, 所以 R 是 3NF 的。但是, 主属性 $Pname$ 传递依赖于码 $(Pname, Mname)$, 因此 R 不是 BCNF 的。当一种零件由多个生产厂家生产时, 零件名与零件号间的联系将多次重复, 带来冗余和操作异常现象。若将 R 分解成:

$R1(Pno, Pname)$ 和 $R2(Pno, Mname)$

就可以解决上述问题, 并且分解后的 $R1$ 和 $R2$ 都属于 BCNF。

5. 4NF(第四范式)

定义: (4NF) 若关系模式 $R \in 1NF$, R 的每个非平凡多值依赖 $X \twoheadrightarrow Y$ 且 $Y \subsetneq X$ 时, X 必含有码, 则关系模式 $R(U, F) \in 4NF$ 。

4NF 限制关系模式的属性间有非平凡且非函数依赖的多值依赖。

注意: 如果只考虑函数依赖, 关系模式规范化程度最高的是 BCNF。如果考虑多值依赖, 关系模式规范化程度最高的是 4NF。

6. 连接依赖 5NF

连接依赖: 当关系模式无损分解为 n 个投影 ($n > 2$) 时会产生一些特殊的情况。下面考虑供应商数据库中 SPJ 关系的一个具体的值, 见图 8-26。

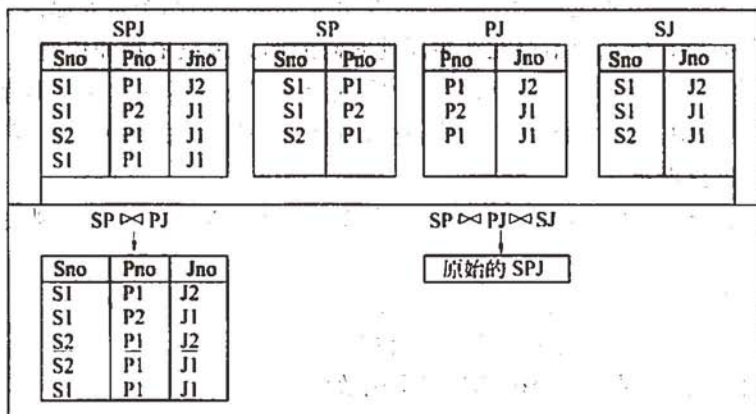


图 8-26 关系 SPJ 是 3 个二元投影的连接

SP 和 PJ 第一次投影连接“ $SP \bowtie PJ$ ”结果比原始 SPJ 关系多了一个元组“S2, P1, J2”,如图 8-26 中带下划线的元组。第二次连接的结果去掉了多余的元组,从而恢复了原始的关系 SPJ。在这种情况下,原始的 SPJ 关系是可三分解的。注意,无论我们选择哪两个投影作为第一次连接,结果都是一样的,尽管每种情况下的中间结果不同。

SPJ 的可三分解性基本上是与时间无关的特性,是关系模式的所有合法值满足的特性。也就是说,这是关系模式满足一个特定的与时间无关的完整性约束。将这种约束简称为 3D(三分解)约束。上述情况就是连接依赖要研究的问题。

连接依赖:如果给定一个关系模式 R ,而 R_1, R_2, \dots, R_n 是 R 的分解,那么称 R 满足连接依赖 $JD^* \{R_1, R_2, \dots, R_n\}$,当且仅当 R 的任何可能出现的合法值都与它在 R_1, R_2, \dots, R_n 上的投影等价。

形式化地说,若 $R = R_1 \cup R_2 \cup \dots \cup R_n$,且 $r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r) \bowtie \dots \bowtie \prod_{R_n}(r)$,则称 R 满足连接依赖 $JD^* \{R_1, R_2, \dots, R_n\}$ 。如果某个 R_i 就是 R 本身,则连接依赖是平凡的。

为了进一步理解连接依赖的概念,可以考虑银行数据库中的贷款(L-no, Bname, C-name, amount)子模式,其中:

贷款号为 L-no 的贷款是由机构名为 Bname 贷出的。

贷款号为 L-no 的贷款是贷给客户名为 C-name 的。

贷款号为 L-no 的贷款金额是 amount。

可以看出这是一个非常直观的逻辑蕴涵连接依赖:

$JD^* ((L-no, Bname), (L-no, C-name), (L-no, amount))$

这个例子说明了连接依赖很直观,符合数据库设计的原则。

定义:一个关系模式 R 是第五范式(也称投影-连接范式 PJNF),当且仅当 R 的每一个非平凡的连接依赖都被 R 的候选码所蕴涵。记作 5NF。

“被 R 的候选码所蕴涵”的含义可通过 SPJ 关系来理解。关系模式 SPJ 并不是 5NF 的,因为它满足一个特定连接依赖,即 3D 约束。这显然没有被其惟一的候选码(该候选码是所有属性的组合)所蕴涵。其区别是,关系模式 SPJ 并不是 5NF,因为它是可三分解的,可三分解并没有为其(Sno, Pno, Jno)候选码所蕴涵。但是将 SPJ 3 分解后,由于 3 个投影 SP、PJ 和 SJ 不包括任何(非平凡的)连接依赖,因此它们都是 5NF 的。

8.4.3 模式分解及分解应具有的特性

1. 分解

定义:关系模式 $R(U, F)$ 的一个分解 ρ 是指 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_n(U_n, F_n)\}$

$F_n\}$), 其中 $U = \bigcup_{i=1}^n U_i$, 并且没有 $U_i \subseteq U_j (1 \leq i, j \leq n)$, F_i 是 F 在 U_i 上的投影, $F_i = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq U_i\}$ 。

对一个给定的模式进行分解, 分解后的模式是否与原来的模式等价有 3 种情况:

- (1) 分解具有无损连接性;
- (2) 分解要保持函数依赖;
- (3) 分解既要无损连接性, 又要保持函数依赖。

2. 无损连接

定义: $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解。若 $R(U, F)$ 的任何一个关系 r 均有 $r = m_\rho(r)$ 成立, 则称分解 ρ 具有无损连接性, 简称无损分解。其中 $m_\rho(r) = \bigotimes_{i=1}^k \pi_{R_i}(r)$ 。

定理: 关系模式 $R(U, F)$ 的一个分解 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2)\}$ 具有无损连接的充分必要的条件是:

$$U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^+ \quad \text{或} \quad U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^+$$

证明略。

3. 保持函数依赖

定义: 设关系模式 $R(U, F)$ 的一个分解 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$, 如果 $F^+ = (\bigcup_{i=1}^k \pi_{R_i}(F^+))$, 则称分解 ρ 保持函数依赖。

4. 判别一个分解的无损连接性和保持函数依赖的算法

【算法 8-1】 判别一个分解的无损连接性。

$\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解, $U = \{A_1, A_2, \dots, A_n\}$, $F = \{FD_1, FD_2, \dots, FD_p\}$, 并设 F 是一个最小依赖集, 记 FD_i 为 $X_i \rightarrow A_{ij}$, 其步骤如下:

- (1) 建立一张 n 列 k 行的表, 每一列对应一个属性, 每一行对应分解中的一个关系模式。若属性 $A_j \in U_i$, 则在 j 列 i 行上填上 a_j , 否则填上 b_{ij} 。
- (2) 对于每一个 FD_i 做如下操作: 找到 X_i 所对应的列中具有相同符号的那些行。考察这些行中 A_{ij} 列的元素, 若其中有 a_{ij} , 则全部改为 a_{ij} , 否则全部改为 $b_{m_{di}}$, m 是这些行的行号最小值。

如果在某次更改后, 有一行成为: a_1, a_2, \dots, a_n , 则算法终止。说明分解 ρ 具有无损连接性, 否则不具有无损连接性。

对 F 中 p 个 FD 逐一进行一次这样的处理, 称为对 F 的一次扫描。

- (3) 比较扫描前后表有无变化, 如有变化, 则返回第(2)步, 否则算法终止。

如果发生循环,那么前次扫描至少应使该表减少一个符号。而表中符号有限,因此,循环必然终止。

【例 8.18】 设有关系模式 $R(U, F)$, 其中 $U = \{A, B, C, D, E\}$, $F = \{AC \rightarrow E, E \rightarrow D, A \rightarrow B, B \rightarrow D\}$, 请判断如下两个分解是否无损连接。

- (1) $\rho_1 = \{R_1(AC), R_2(ED), R_3(AB)\}$
- (2) $\rho_2 = \{R_1(ABC), R_2(ED), R_3(ACE)\}$

解: (1) 判断 ρ_1 是否无损。根据【算法 8-1】, 构造一个二维矩阵, 其元素如表 8-3 所示。

表 8-3 与 ρ_1 有关的构造矩阵元素

属性 模式	A	B	C	D	E
$R_1(AC)$	a_1	b_{12}	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{34}	b_{35}

根据 $A \rightarrow B$ 对表 8-3 进行处理。由于属性列 A 的第一行和第三行相同(均为 a_1), 所以将属性列 B 中的 b_{12} 改为同一符号 a_2 。修改后的情况如表 8-4 所示。

表 8-4 修改后的矩阵元素

属性 模式	A	B	C	D	E
$R_1(AC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{34}	b_{35}

根据 F 中的 $B \rightarrow D$ 对表 8-4 进行处理。由于属性列 B 的第一行和第三行相同(均为 a_2), 所以将属性列 D 中的 b_{14} 和 b_{34} 改为同一符号 b_{14} , 取行号最小值(因为属性列 D 中的第一行和第三行没有 a_4), 如表 8-5 所示。

表 8-5 $B \rightarrow D$ 处理后的情况

属性 模式	A	B	C	D	E
$R_1(AC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(AB)$	a_1	a_2	b_{33}	b_{14}	b_{35}

反复检查函数依赖集 F , 已无法修改上表, 故分解 ρ_1 是有损的。

(2) 判断 ρ_2 是否无损。根据算法 8-1, 构造一个二维矩阵如表 8-6 所示。 $F = \{AC \rightarrow E, E \rightarrow D, A \rightarrow B, B \rightarrow D\}$

表 8-6 与 ρ_1 有关的元素

属性 模式	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	b_{14}	b_{15}
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{32}	a_3	b_{34}	a_5

根据 F 中的 $AC \rightarrow E$, AC 属性列的第一行和第三行相同 (均为 a_1 和 a_3), 所以将属性列 E 中的 b_{15} 改为同一符号 a_5 , 如表 8-7 所示。

表 8-7 $AC \rightarrow E$ 处理后的情况

属性 模式	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	b_{14}	a_5
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{23}	a_3	b_{34}	a_5

根据 $E \rightarrow D$, E 属性列的第一行、第二行和第三行相同 (均为 a_5), 所以将属性列 D 中的 b_{14} 和 b_{34} 改为同一符号 a_4 , 如表 8-8 所示。

表 8-8 $E \rightarrow D$ 处理后的情况

属性 模式	A	B	C	D	E
$R_1(ABC)$	a_1	a_2	a_3	a_4	a_5
$R_2(ED)$	b_{21}	b_{22}	b_{23}	a_4	a_5
$R_3(ACE)$	a_1	b_{23}	a_3	a_4	a_5

从表 8-8 可以看出第一行全为 a , 故分解 ρ_2 是无损连接。

【算法 8-2】 转换成 3NF 的保持函数依赖的分解。

$\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 是关系模式 $R(U, F)$ 的一个分解, $U = \{A_1, A_2, \dots, A_n\}$, $F = \{FD_1, FD_2, \dots, FD_p\}$, 并设 F 是一个最小依赖集, 记 FD_i 为 $X_i \rightarrow A_{ij}$ 。其步骤如下:

(1) 对 $R(U, F)$ 的函数依赖集 F 进行极小化处理(处理后的结果仍记为 F)。

(2) 找出不在 F 中出现的属性, 将这样的属性构成一个关系模式。把这些属性从 U 中去掉, 剩余的属性仍记为 U 。

(3) 若有 $X \rightarrow A \in F$, 且 $XA = U$, 则 $\rho = \{R\}$, 算法终止。

(4) 否则, 对 F 按具有相同左部的原则分组(假定分为 k 组), 每一组函数依赖 F_i 所涉及的全部属性形成一个属性集 U_i 。若 $U_i \subseteq U_j (i \neq j)$ 就去掉 U_i 。由于经过了步骤(2), 故 $U = \bigcup_{i=1}^k U_i$, 于是 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$ 构成 $R(U, F)$ 的一个保持函数依赖的分解。并且, 每个 $R_i(U_i, F_i)$ 均属于 3NF 且保持函数依赖。

【例 8.19】 设有关系模式 $R(U, F)$, 其中 $U = \{C, T, H, I, S, G\}$, $F = \{CS \rightarrow G, C \rightarrow T, TH \rightarrow I, HI \rightarrow C, HS \rightarrow I\}$, 将其分解成 3NF 并保持函数依赖。

解: 根据算法 8-2 求解如下。

(1) F 已为最小函数依赖集, 所以转(2);

(2) R 中的所有属性均在 F 中出现, 所以转(3);

(3) 对 F 按具有相同左部的原则分组为 $R_1 = CSG, R_2 = CT, R_3 = THI, R_4 = HIC$ 和 $R_5 = HSI$ 。

所以, 关系模式 R 分解成 3NF 并保持函数依赖的分解为 $\rho = \{R_1(CSG), R_2(CT), R_3(THI), R_4(HIC), R_5(HSI)\}$ 。

【算法 8-3】 将一个关系模式转换成 3NF, 使它既具有无损连接又保持函数依赖的分解。

输入: 关系模式 R 和 R 的最小函数依赖集 F 。

输出: $R(U, F)$ 的一个分解 $\rho = \{R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k)\}$, R_i 为 3NF, 且 ρ 具有无损连接又保持函数依赖的分解。

操作步骤如下:

(1) 根据算法 8-2 求出保持依赖的分解 $\rho = \{R_1, R_2, \dots, R_n\}$;

(2) 判断分解 ρ 是否具有无损连接性。若有, 转(4);

(3) 令 $\rho = \rho \cup \{X\}$, 其中 X 是 R 的码;

(4) 输出 ρ 。

【例 8.20】 将例 8.19 的关系模式 $R(U, F)$ 分解成 3NF, 使分解 ρ 具有无损连接又保持函数依赖。

解: 根据算法 8-3 求解如下。

(1) 据例 8.19 得 3NF 保持函数依赖的分解如下:

$$\rho = \{R_1(CSG), R_2(CT), R_3(THI), R_4(HIC), R_5(HSI)\}$$

(2) 根据算法 8-1, 构造一个二维矩阵, 其元素如表 8-9 所示。

表 8-9 与 p 有关的元素

属性 模式	C	T	H	I	S	G
R_1 (CSG)	a_1	b_{12}	b_{13}	b_{14}	a_5	a_6
R_2 (CT)	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
R_3 (THI)	b_{31}	a_2	a_3	a_4	b_{35}	b_{36}
R_4 (HIC)	a_1	b_{42}	a_3	a_4	b_{45}	b_{46}
R_5 (HSI)	b_{51}	b_{52}	a_3	a_4	a_5	b_{56}

根据 F 中的 $C \rightarrow T$ 对表 8-9 进行处理, 由于属性列 C 的第一行、第二行及第四行相同 (均为 a_1), 所以将属性列 T 中的 b_{12} 和 b_{42} 改为同一符号 a_2 。又根据 $HI \rightarrow C$ 将属性列 C 中的 b_{31} 和 b_{51} 改为同一符号 a_1 , 如表 8-10 所示。

表 8-10 修改后的矩阵元素

属性 模式	C	T	H	I	S	G
R_1 (CSG)	a_1	a_2	b_{13}	b_{14}	a_5	a_6
R_2 (CT)	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
R_3 (THI)	a_1	a_2	a_3	a_4	b_{35}	b_{36}
R_4 (HIC)	a_1	a_2	a_3	a_4	b_{45}	b_{46}
R_5 (HSI)	a_1	b_{52}	a_3	a_4	a_5	b_{56}

根据 F 中的 $CS \rightarrow G$ 对表 8-10 进行处理, 由于属性列 CS 的第一行和第五行相同 (均为 a_1 和 a_5), 所以将属性列 G 中的 b_{56} 改为同一符号 a_6 。又根据 $C \rightarrow T$ 将属性列 T 中的 b_{52} 改为同一符号 a_2 , 如表 8-11 所示。

表 8-11 再次修改后的矩阵元素

属性 模式	C	T	H	I	S	G
R_1 (CSG)	a_1	a_2	b_{13}	b_{14}	a_5	a_6
R_2 (CT)	a_1	a_2	b_{23}	b_{24}	b_{25}	b_{26}
R_3 (THI)	a_1	a_2	a_3	a_4	b_{35}	b_{36}
R_4 (HIC)	a_1	a_2	a_3	a_4	b_{45}	b_{46}
R_5 (HSI)	a_1	a_2	a_3	a_4	a_5	a_6

从上述结果可见,已找到一行(第五行)为全 a ,所以分解 ρ 是无损连接并保持函数依赖的。

【算法 8-4】 将关系模式转换成 BCNF,使它具有无损连接的分解。

输入: 关系模式 R 和函数依赖集 F 。

输出: $R(U, F)$ 的一个分解 $\rho = \{ R_1(U_1, F_1), R_2(U_2, F_2), \dots, R_k(U_k, F_k) \}$, R_i 为 BCNF,且 ρ 具有无损连接的分解。

操作步骤如下:

(1) 令 $\rho = \{ R \}$;

(2) 若 ρ 中的所有模式都是 BCNF,则转(4);

(3) 若 ρ 中有一个关系模式 R_i 不是 BCNF,则 R_i 中必能找到一个函数依赖 $X \rightarrow A$,且 X 不是 R_i 的候选码, A 不属于 X 。设 $R_{i1}(XA), R_{i2}(R_i - A)$,用分解 $\{R_{i1}, R_{i2}\}$ 代替 R_i ,转(2);

(4) 输出 ρ 。

【例 8.21】 设有关系模式 $R(U, F)$,其中 $U = \{C, T, H, I, S, G\}$, $F = \{CS \rightarrow G, C \rightarrow T, TH \rightarrow I, HI \rightarrow C, HS \rightarrow I\}$,将其无损连接地分解成 BCNF。

解: 检查 F 发现, H 和 S 只出现在函数依赖的左部,所以为 L 类属性。又因为 F 中无 NLR 属性,所以 R 上只有一个候选码 HS 。

(1) 令 $\rho = \{ R(U, F) \}$;

(2) ρ 中并非所有的模式都是 BCNF,转(3);

(3) 考虑 $CS \rightarrow G$,这个函数依赖不满足 BCNF 条件(CS 不包含候选码 HS),所以将其分解成 $R_1(CSG)$ 和 $R_2(CTHIS)$ 。计算 R_1 和 R_2 的最小函数依赖集分别为 $F_1 = \{ CS \rightarrow G \}$ 和 $F_2 = \{ C \rightarrow T, TH \rightarrow I, HI \rightarrow C, HS \rightarrow I \}$ 。 R_2 的候选码为 HS 。

因为 R_1 已是 BCNF,而 R_2 的 F_2 中存在不为码的决定因素 $HI \rightarrow C$,所以 R_2 不属于 BCNF,应当进一步分解 R_2 。

分解 R_2 : 考虑 $C \rightarrow T$,将其分解成 $R_{21}(CT)$ 和 $R_{22}(CHIS)$ 。计算 R_{21} 和 R_{22} 的最小函数依赖集分别为 $F_{21} = \{ C \rightarrow T \}$ 和 $F_{22} = \{ CH \rightarrow I, HI \rightarrow C, HS \rightarrow I \}$ 。

因为 $C \rightarrow T, TH \rightarrow I$ 。所以在 F_{22} 中,有 $CH \rightarrow I$ 。 R_{22} 的候选关键字为 HS 。

因为 R_{21} 已是 BCNF, R_{22} 不是 BCNF。所以进一步分解 R_{22} 即可。

分解 R_{22} : 考虑 $CH \rightarrow I$,将其分解成 $R_{221}(CHI)$ 和 $R_{222}(CHS)$ 。计算 R_{221} 和 R_{222} 的最小函数依赖集分别为 $F_{221} = \{ CH \rightarrow I, HI \rightarrow C \}$ 和 $F_{222} = \{ HS \rightarrow C \}$ 。

因为 R_{221} 和 R_{222} 已是 BCNF。所以将 R 分解后, $\rho = \{ R_1(CSG), R_{21}(CT), R_{221}(CHI), R_{222}(CHS) \}$ 。

第9章 SQL 语言

SQL(structured query language)早已确立了自己作为关系数据库标准语言的地位,目前已成为数据库的主流语言。它不仅包含数据查询功能,还包括插入、删除、更新和数据定义功能。一个 SQL 数据库是表的汇集,它用一个或多个 SQL 模式定义。基本表是实际存储在数据库中的表,而视图是由若干个基本表或其他视图导出的表,称为“虚表”。作为 SQL 的用户可以是应用程序,也可以是终端用户。

9.1 数据库语言

9.1.1 数据库语言概述

任何一个数据库系统都应向用户提供一种数据库语言,包括数据定义和数据操纵子语言。SQL 语言是集数据定义和数据操纵为一体的典型数据库语言。数据库语言与数据模型密切相关,基于不同的数据模型,数据库语言也不同。目前的关系数据库系统产品都提供 SQL 语言作为标准数据库语言。

数据定义子语言用来定义数据库模式,简记为 DDL。DDL 包括数据库模式定义,数据库存储结构和存取方法定义,以及数据库模式的修改和删除功能。数据定义子语言的处理程序分为数据库模式定义处理程序,数据库存储结构和存取方法定义处理程序。数据库模式定义处理程序接收用 DDL 表示的数据模式定义,将其转变为内部表示形式,存储到数据字典中。数据库存储结构和存取方法定义处理程序接收数据库系统存储结构和存取方法定义,在存储设备上创建相关的数据库文件,建立物理数据库。

数据操纵子语言用来表示用户对数据库的操作请求。通常,数据操纵语言能表示如下数据库操作:

- 查询数据库中的信息;
- 向数据库插入新的信息;
- 从数据库删除信息;
- 修改数据库中的信息。

数据库操纵语言分为过程性和非过程性两种。过程性语言要求用户既要说明需要数据库中的什么数据,也要说明怎样搜索这些数据。非过程性语言只要求用户说明需要数

数据库中的什么数据,不需要说明怎样搜索这些数据。非过程性语言比过程性语言易学、易懂,但是非过程性语言产生的处理程序代码要比过程性语言产生的代码效率低。这个问题可以通过查询优化来解决。数据操纵语言的核心是数据的查询,所以,有时人们也把数据操纵语言称为数据查询语言。严格地说这种说法是不确切的。

9.1.2 数据库语言的分类

SQL 语言可以作为独立语言在终端上以交互方式使用,也可作为程序设计的子语言使用。即嵌入到高级语言中使用。这种方式下使用的 SQL 称为嵌入式 SQL,嵌入 SQL 的高级语言称为宿主语言。

在 DBMS 中,对宿主型数据库语言 SQL 采用两种方法处理,第一种方法是采用预编译,第二种方法是修改和扩充主语言,使之能处理 SQL 语句。

目前采用最多的是预编译的方法。该方法由 DBMS 的预处理程序对源程序进行扫描,识别出 SQL 语句,把它们转换为主语言调用语句,使主语言的编译程序能识别它,最后由主语言的编译程序将整个源程序编译成目标码。综上所述,若采用第一种方法,则必须区分主语言中嵌入的 SQL 语句,及主语言和 SQL 间的通信问题。后续章节将分别加以介绍。

9.2 SQL 概述

与自然语言的方言一样,存在许多不同版本的 SQL。目前主要有 3 个标准:ANSI (美国国家标准机构)SQL;对 ANSI SQL 进行修改后从 1992 年开始采用的标准称之为 SQL-92 或 SQL2;最近的 SQL-99 标准也称 SQL3 标准。SQL-99 从 SQL-92 扩充而来,并增加了对对象关系特征和许多其他新的功能。其次,各个厂家也提供不同版本的 SQL。这些版本不仅都包含原始的 ANSI 标准,而且在很大程度上都支持 SQL-92 标准,并在 SQL-92 的基础上做了修改和扩展,包括部分 SQL-99 标准的内容。

9.2.1 SQL 语句的特征

尽管人们习惯性地称 SQL 是一种“查询语言”,但实际上,它的功能远非查询信息这么简单,它还要包括数据查询(query)、数据操纵(manipulation)、数据定义(definition)和数据控制(control)功能,是一种通用的、功能强大的关系数据库语言。

1. SQL 的特点

(1) 综合统一:非关系模型的数据语言分为模式定义语言和数据操纵语言。其缺点是,当要修改模式时,必须停止现有数据库的运行,转储数据,修改模式,编译后再重装数

数据库。SQL 集数据定义、数据操纵和数据控制功能于一体，语言风格统一，可独立完成数据库生命周期的所有活动。

(2) 高度非过程化：非关系数据模型的数据操纵语言是面向过程的，若要完成某项请求时，必须指定存储路径。而 SQL 语言是高度非过程化语言，当进行数据操作时，只要指出“做什么”，无须指出“怎么做”，存储路径对用户来说是透明的，提高了数据的独立性。

(3) 面向集合的操作方式：非关系数据模型采用的是面向记录的操作方式，操作对象是记录。而 SQL 语言采用面向集合的操作方式，其操作对象和查找结果可以是元组的集合。

(4) 两种使用方式：第一种方式，用户可以在终端键盘上键入 SQL 命令，对数据库进行操作，故称之为自含式语言。第二种方式，将 SQL 语言嵌入到高级语言程序中，所以 SQL 又是嵌入式语言。

(5) 语言简洁、易学易用：SQL 语言功能极强，完成核心功能只用了 9 个动词，包括如下四类：

- 数据查询：SELECT。
- 数据定义：CREATE、DROP、ALTER。
- 数据操纵：INSERT、UPDATE、DELETE。
- 数据控制：GRANT、REVOKE。

2. SQL 支持三级模式结构

SQL 语言支持关系数据库的三级模式结构，其中视图对应外模式，基本表对应模式，存储文件对应内模式。具体结构见图 9-1。

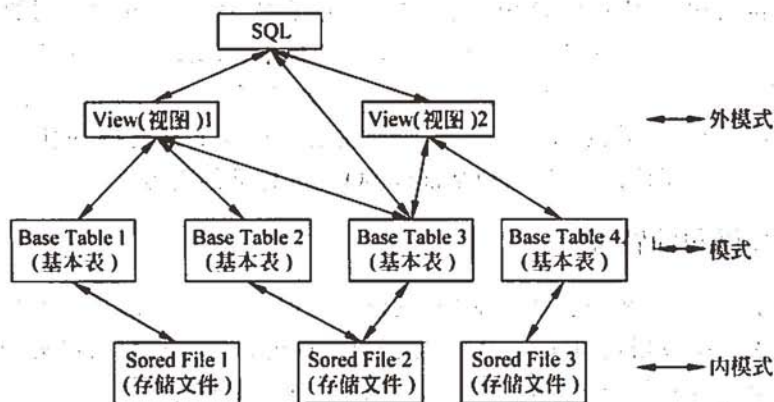


图 9-1 关系数据库的三级模式结构

9.2.2 SQL 的基本组成

SQL 由以下几个部分组成。

数据定义语言(DDL): SQL DDL 提供定义关系模式和视图、删除关系和视图以及修改关系模式的命令。

交互式数据操纵语言(DML): SQL DML 提供查询、插入、删除和修改的命令。

事务控制(transaction control): SQL 提供定义事务开始和结束的命令。

嵌入式 SQL 和动态 SQL(embedded SQL and dynamic SQL): 用于嵌入到某种通用的高级语言(C、C++、Java、PL/I、Cobol 和 VB 等)中混合编程。其中 SQL 负责操纵数据库,高级语言负责控制程序流程。

完整性(integrity): SQL DDL 包括定义数据库中的数据必须满足完整性约束条件的命令,对于破坏完整性约束条件的更新将被禁止。

权限管理(authorization): SQL DDL 中包括设定关系和视图访问权限的命令。

本章主要介绍基本的 DDL 和 DML 以及嵌入式 SQL 和动态 SQL。后面将介绍数据库连接技术,如 ODBC 等。

9.3 数据库定义

9.3.1 创建表

创建表(CREATE TABLE)语句格式:

```
CREATE TABLE <表名>(<列名><数据类型>[列级完整性约束条件]  
[,<列名><数据类型>[列级完整性约束条件]]...  
[,<表级完整性约束条件>];
```

列级完整性约束条件有 NULL(空)和 UNIQUE(取值惟一)。如 NOT NULL、UNIQUE 表示取值惟一,不能取空值。

【例 9.1】 建立一个供应商和零件数据库。其中“供应商”表 S(Sno, Sname, Status, City)的属性分别表示供应商代码、供应商名、供应商状态及供应商所在城市。“零件”表 P(Pno, Pname, Color, Weight, City)的属性分别表示零件号、零件名、颜色、重量及产地。其中,数据库要满足如下要求:

- (1) 供应商代码不能为空,且值是惟一的,供应商的名也是惟一的。
- (2) 零件号不能为空,且值是惟一的。零件名不能为空。
- (3) 一个供应商可以供应多个零件,而一个零件可以由多个供应商供应。

分析：根据题意供应商和零件要分别建立一个关系模式。供应商和零件之间是一个多对多的联系。在关系数据库中，多对多联系必须生成一个关系模式，而该模式的码是该联系两端实体的码加上联系的属性构成的。若该联系名为 SP，那么关系模式为 SP(Sno, Pno, Qty)，其中 Qty 表示零件的数量。

根据上述分析，用 SQL 建立一个供应商、零件数据库如下：

```
CREATE TABLE S(Sno CHAR(5) NOT NULL UNIQUE,
```

```
    Sname CHAR(30) UNIQUE,
```

```
    Status CHAR(8) ,
```

```
    City CHAR(20)
```

```
    PRIMARY KEY(Sno));
```

```
CREATE TABLE P(Pno CHAR(6),
```

```
    Pname CHAR(30) NOT NULL,
```

```
    Color CHAR(8) ,
```

```
    Weight NUMERIC(6,2),
```

```
    City CHAR(20)
```

```
    PRIMARY KEY(Pno));
```

```
CREATE TABLE SP(Sno CHAR(5),
```

```
    Pno CHAR(6),
```

```
    Status CHAR(8) ,
```

```
    Qty NUMERIC(9),
```

```
    PRIMARY KEY(Sno,Pno),
```

```
    FOREIGN KEY(Sno) REFERENCES S(Sno),
```

```
    FOREIGN KEY(Pno) REFERENCES P(Pno));
```

从上述定义可以看出，“Sno CHAR(5) NOT NULL UNIQUE”语句定义了 Sno 的列级完整性约束条件，取值惟一，且不能取空值。

需要说明的是：

(1) PRIMARY KEY(Sno) 已经定义了 Sno 为主码，所以，“Sno CHAR(5) NOT NULL UNIQUE”语句中的“NOT NULL UNIQUE”可以省略。

(2) FOREIGN KEY(Sno) REFERENCES S(Sno) 定义了 SP 关系中的 Sno 为外码，其取值必须来自 S 关系的 Sno 域。同理，SP 关系中 Pno 也定义为外码。

9.3.2 修改表和删除表

1. 修改表(ALTER TABLE)

语句格式：

```
ALTER TABLE <表名>[ADD<新列名><数据类型>[完整性约束条件]]  
[DROP<完整性约束名>]  
[MODIFY <列名><数据类型>];
```

例如,在“供应商”表 S 中增加 Zap“邮政编码”属性可用如下语句:

```
ALTER TABLE S ADD Zap CHAR(6);
```

注意,不论基本表中原来是否已有数据,新增加的列一律为空。

又如,将 Status 字段改为整型可用如下语句:

```
ALTER TABLE S MODIFY Status INT;
```

2. 删除表(DROP TABLE)

语句格式:

```
DROP TABLE <表名>
```

例如,执行 DROP TABLE Student 后,关系 Student 不再是数据库模式的一部分,关系中的元组也无法访问。

9.3.3 定义和删除索引

数据库中的索引与书籍中的索引类似。在一本书中,利用索引可以快速查找所需信息,无须阅读整本书。在数据库中,索引使数据库程序无须对整个表进行扫描,就可以从中找到所需数据。书中的索引是一个词语列表,其中注明了包含各个词的页码。而数据库中的索引是某个表中一列或者若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。

索引的作用如下:

- 通过创建惟一的索引,可以保证数据记录的惟一性。
- 可以大大加快数据检索速度。
- 可以加速表与表之间的连接,这一点在实现数据的参照完整性方面有特别的意义。
- 在使用 ORDER BY 和 GROUP BY 子句检索数据时,可以显著减少查询中分组和排序的时间。
- 使用索引可以在检索数据的过程中使用优化隐藏器,提高系统性能。

1. 聚集索引和非聚集索引

聚集索引对表的物理数据页中的数据按列进行排序,然后再重新存储到磁盘上,即聚集索引与数据是混为一体的,它的叶节点中存储的是实际的数据。

非聚集索引具有完全独立于数据行的结构,使用非聚集索引不用将物理数据页中的数据按列排序。非聚集索引的叶节点存储的是组成非聚集索引的关键字值和行定位器。

2. 建立索引

语句格式:

```
CREATE [UNIQUE][CLUSTER]INDEX <索引名>  
ON <表名>(<列名>[<次序>][,<列名>[<次序>]]...);
```

说明:

- (1) 次序: 可选 ASC(升序)或 DSC(降序), 默认值为 ASC。
- (2) UNIQUE: 表明此索引的每一个索引值只对应惟一的数据记录。
- (3) CLUSTER: 表明要建立的索引是聚簇索引, 意为索引项的顺序是与表中记录的物理顺序一致的索引组织。

【例 9.2】 假设供应销售数据库中有供应商 S、零件 P、工程项目 J 和供销情况 SPJ 关系, 希望建立 4 个索引。其中供应商 S 中 Sno 按升序建立索引, 零件 P 中 Pno 按升序建立索引, 工程项目 J 中 Jno 按升序建立索引, 对供销情况 SPJ 中的 Sno 按升序, Pno 按降序, Jno 按升序建立索引。

解: 根据题意建立索引如下:

```
CREATE UNIQUE INDEX S-SNO ON S(Sno);  
CREATE UNIQUE INDEX P-PNO ON P(Pno);  
CREATE UNIQUE INDEX J-JNO ON J(Jno);  
CREATE UNIQUE INDEX SPJ-NO ON SPJ(Sno ASC, Pno DESC, JNO ASC)。
```

3. 删除索引

语句格式:

```
DROP INDEX <索引名>
```

例如, 执行 DROP INDEX StudentIndex 后, 索引 StudentIndex 不再是数据库模式的一部分。

9.3.4 定义、删除、更新视图

视图是从一个或者多个表或视图中导出的表, 其结构和数据是建立在对表的查询基础上的。和真实的表一样, 视图也包括几个被定义的数据列和多个数据行。但从本质上讲, 这些数据列和数据行来源于其所引用的表。因此, 视图不是真实存在的基础表, 而是一个虚拟表, 视图所对应的数据并不实际地按视图结构存储在数据库中, 而是存储在视图所引用的表中。

使用视图的优点和作用如下:

- 使用视图可以集中数据、简化和定制不同用户对数据库的不同数据要求。

- 使用视图可以屏蔽数据的复杂性,用户不必了解数据库的结构,就可以方便地使用和管理数据,简化数据权限管理和重新组织数据以便输出到其他应用程序中。
- 视图可以使用户只关心其感兴趣的某些特定数据及其所负责的特定任务,而那些不需要的或者无用的数据则不在视图中显示。
- 视图大大地简化了用户对数据的操作。
- 视图可以让不同的用户以不同的方式看到不同或者相同的数据集。
- 在某些情况下,由于表中数据量太大,因此在设计表时常将表进行水平或者垂直分割,但表结构的变化对应用程序会产生不良的影响。
- 视图提供了一个简单而有效的安全机制。

1. 视图的创建

语句格式:

```
CREATE VIEW 视图名 (列表名)
AS SELECT 查询子句
[WITH CHECK OPTION];
```

注意,创建视图时必须遵循如下规定:

(1) 子查询可以是任意复杂的 SELECT 语句,但通常不允许含有 order by 子句和 DISTINCT 短语。

(2) WITH CHECK OPTION 表示执行 UPDATE, INSERT 或 DELETE 操作时保证更新、插入或删除的行满足视图定义中的谓词条件(即子查询中的条件表达式)。

(3) 组成视图的属性列名或者全部省略或者全部指定。如果省略属性列名,则隐含该视图由 SELECT 子查询目标列的主属性组成。

为了便于后续内容的介绍,假定教学数据库中的关系模式如下所示。

学生关系模式: Students(Sno, Sname, Sex, SD, Sage, SAdd)
 课程关系模式: C(Cno, Cname, Pcn)
 学生选课关系模式: SC(Sno, Cno, Grade)

【例 9.3】 建立“计算机系”(CS 表示计算机系)学生的视图,并要求执行修改、插入操作时能够保证该视图只有计算机系的学生。

```
CREATE VIEW CS-STUDENT
AS SELECT Sno, Sname, Sage, Sex
FROM Students
WHERE SD='CS'
```

WITH CHECK OPTION;

由于 CS-STUDENT 视图使用了 WITH CHECK OPTION 子句,因此,对该视图进行修改或插入操作时 DBMS 会自动加上 SD='CS'的条件,保证该视图只有计算机系的学生。

2. 视图的删除

DROP VIEW 视图名

例如,DROP VIEW CS-STUDENT 将删除视图 CS-STUDENT。

9.4 数据操作

SQL 的数据操纵功能包括 SELECT(查询)、INSERT(插入)、DELETE(删除)和 UPDATE(修改)4 条语句。SQL 语言对数据库的操作十分灵活方便,原因在于 SELECT 语句中的成分丰富多样,有许多可选形式,尤其是目标列和条件表达式。

9.4.1 Select 基本结构

数据库查询是数据库的核心操作,SQL 语言提供了 SELECT 语句,用于执行数据库的查询。

语句格式:

```
SELECT [ALL|DISTINCT]<目标列表达式>[,<目标列表达式>]...  
FROM <表名或视图名>[,<表名或视图名>]  
[WHERE <条件表达式>]  
[GROUP BY <列名 1>[HAVING<条件表达式>]]  
[ORDER BY <列名 2>[ASC|DESC]...]
```

- SQL 查询中的子句顺序: SELECT、FROM、WHERE、GROUP BY、HAVING 和 ORDER BY。其中 SELECT 和 FROM 是必须的,而 HAVING 子句只能与 GROUP BY 搭配起来使用。
- SELECT 子句对应的是关系代数中的投影运算,用来列出查询结果中的属性。其输出可以是列名、表达式和集函数(AVG、COUNT、MAX、MIN 及 SUM)。DISTINCT 选项可以保证查询的结果集中不存在重复元组。
- FROM 子句对应的是关系代数中的笛卡儿积,它列出的是表达式求值过程中须扫描的关系。在 FROM 子句中出现多个基本表或视图时,系统首先执行笛卡儿积操作。
- WHERE 子句对应的是关系代数中的选择谓词。WHERE 子句的条件表达式中可以使用的运算符如表 9-1 所示。

表 9-1 WHERE 子句的条件表达式中可以使用的运算符

运 算 符		含 义	运 算 符		含 义
集 合 成 员 运 算 符	IN	在集合中	算 术 运 算 符	>	大于
	NOT IN	不在集合中		≥	大于等于
字 符 串 匹 配 运 算 符	LIKE	与_和%进行单 个多个字符匹配		<	小于
				≤	小于等于
				=	等于
	≠	不等于			
空 值 比 较 运 算 符	IS NULL	为空	逻辑运算符	AND	与
	IS NOT NULL	不能为空		OR	或
				NOT	非

一个典型的 SQL 查询具有如下形式：

```
SELECT A1,A2,...,An
FROM R1,R2,...,RM
WHERE P
```

对应的关系代数表达式为 $\prod_{A_1,A_2,\dots,A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$ 。

说明 1：SELECT 查询中没有带全程量词 \forall ，但可以将带全程量词的谓词转换成等价的带有存在量词的谓词，其形式如下：

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

说明 2：SELECT 查询中没有逻辑蕴涵，但可以利用谓词演算将逻辑蕴涵谓词进行等价的转换，其形式如下：

$$p \rightarrow q \equiv \neg p \vee q$$

9.4.2 简单查询

最简单的 SQL 查询是找出关系中满足特定条件的元组，这些查询与关系代数中的选择操作类似。简单查询只需要使用 3 个保留字 SELECT、FROM 和 WHERE。

【例 9.4】 查询学生-课程数据库中计算机系学生的学号、姓名及年龄。

```
SELECT Sno,Sname,Sage
FROM Students
WHERE SD='CS';
```

注意，为了便于理解查询语句的结构，写 SQL 语句时通常要将保留字如 FROM 或 WHERE 作为每一行的开头。但是，如果一个查询或子查询非常短，可以直接将它们写在一行上，这种风格使得查询语句很紧凑，具有很好的可读性。如上例也可写成如下形式：

```
SELECT Sno, Sname, Sage FROM Students WHERE SD='CS';
```

【例 9.5】 查询数学系全体学生的详细信息。

```
SELECT * FROM Students WHERE SD='MS';
```

【例 9.6】 查询学生的出生年份。

```
SELECT Sno, 2004-Sage FROM Students;
```

9.4.3 连接查询

若查询涉及两个以上的表,则称为连接查询。

【例 9.7】 检索选修了课程号为“C1”的学生号和学生姓名可用连接查询和嵌套查询实现,实现方法如下:

```
SELECT Sno, Sname  
FROM Students, SC  
WHERE Students. Sno=SC. Sno AND SC. Cno='C1'
```

【例 9.8】 检索选修课程名为“MS”的学生号和学生姓名可用连接查询和嵌套查询实现,实现方法如下:

```
SELECT Sno, Sname  
FROM Students, SC, C  
WHERE Students. Sno=SC. Sno AND SC. Cno=C. Cno AND C. Cname='MS'
```

【例 9.9】 检索至少选修了课程号为“C1”和“C3”的学生号,实现方法如下:

```
SELECT Sno  
FROM SC SCX, SC SCY  
WHERE SCX. Sno=SCY. Sno AND SCX. Cno='C1' AND SCY. Cno='C3'
```

9.4.4 子查询与聚集函数

1. 子查询

子查询也称嵌套查询。嵌套查询是指一个 SELECT-FROM-WHERE 查询块可以嵌入另一个查询块之中。SQL 中允许多重嵌套。

【例 9.10】 采用嵌套查询来实现例 9.8。

```
SELECT Sno, Sname  
FROM Students  
WHERE Sno IN  
(SELECT Sno
```



```

FROM SC
WHERE Cno IN
    (SELECT Cno
     FROM C
     WHERE Cname='MS'))

```

2. 聚集函数

聚集函数是以一个值的集合为输入,返回单个值的函数。SQL 提供了 5 个预定义的集函数:平均值 AVG、最小值 MIN、最大值 MAX、求和 SUM 以及计数 COUNT。如表 9-2 所示。

表 9-2 集函数的功能

集函数名	功 能
COUNT([DISTINCT ALL] *)	统计元组个数
COUNT([DISTINCT ALL]<列名>)	统计一列中值的个数
SUM([DISTINCT ALL]<列名>)	计算一列(该列应为数值型)中值的总和
AVG([DISTINCT ALL]<列名>)	计算一列(该列应为数值型)值的平均值
MAX([DISTINCT ALL]<列名>)	求一列值的最大值
MIN([DISTINCT ALL]<列名>)	求一列值的最小值

使用 ANY 和 ALL 谓词必须同时使用比较运算符,其含义及等价的转换关系见表 9-3。用集函数实现子查询通常要比直接用 ALL 或 ANY 查询效率要高。

表 9-3 ANY 和 ALL 谓词的含义及等价的转换关系

谓词	语 义	等价转换关系
>ANY	大于子查询结果中的某个值	>MIN
>ALL	大于子查询结果中的所有值	>MAX
<ANY	小于子查询结果中的某个值	<MAX
<ALL	小于子查询结果中的所有值	<MIN
>=ANY	大于等于子查询结果中的某个值	>=MIN
>=ALL	大于等于子查询结果中的所有值	>=MAX
<=ANY	小于等于子查询结果中的某个值	<=MAX
<=ALL	小于等于子查询结果中的所有值	<=MIN
<>ANY	不等于子查询结果中的某个值	--
<>ALL	不等于子查询结果中的任何一个值	NOT IN
=ANY	等于子查询结果中的某个值	IN
=ALL	等于子查询结果中的所有值	--

【例 9.11】 查询课程 C1 的最高分、最低分以及高低分之间的差距。

```
SELECT MAX(G),MIN(G),MAX(G)-MIN(G)
FROM Sc
WHERE Cno='C1'
```

【例 9.12】 查询其他系比计算机系 CS 所有学生年龄都要小的学生姓名及年龄。

方法 1(用 ALL 谓词):

```
SELECT Sname,Sage
FROM Students
WHERE Sage< ALL
      (SELECT Sage
       FROM Students
       WHERE SD='CS')
AND SD<>'CS'
```

方法 2(用 MIN 集函数): 从说明 4 所述的等价转换关系表 9-3 中可见,“<ALL”可用“<MIN”代换。

```
SELECT Sname,Sage
FROM Students
WHERE Sage<
      (SELECT MIN (Sage)
       FROM Students
       WHERE SD='CS' )
AND SD<>'CS'
```

方法 2 实际上是找出计算机系年龄最小的学生的年龄,只要其他系的学生年龄比这个年龄小,那么就应在结果集中。

【例 9.13】 查询其他系比计算机系某一学生年龄小的学生姓名及年龄。

方法 1(用 ANY 谓词):

```
SELECT Sname,Sage
FROM Students
WHERE Sage< ANY
      (SELECT Sage
       FROM Students
       WHERE SD='CS')
AND SD<>'CS'
```

方法 2(用 MAX 集函数): 从说明 4 所述的等价转换关系表 9-3 中可见,“<ANY”可

用“<MAX”代换。

```
SELECT Sname,Sage
FROM Students
WHERE Sage<
      (SELECT MAX (Sage)
       FROM Students
        WHERE SD='CS' )
AND SD<>'CS'
```

方法 2 实际上是找出计算机系年龄最大的学生的年龄,只要其他系的学生年龄比这个年龄大,那么就应在结果集中。

9.4.5 分组查询

1. GROUP BY 子句

在 WHERE 子句后面加上 GROUP BY 子句可以对元组进行分组。保留字 GROUP BY 后面跟着一个分组属性列表。最简单的情况是 FROM 子句后面只有一个关系,根据分组属性对其元组进行分组。SELECT 子句中使用的聚集操作符仅用在每个分组上。

【例 9.14】 针对学生数据库中的 SC 关系,查询每个学生的平均成绩。

```
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno
```

该语句是将 SC 关系的元组重新组织,并进行分组,使得学号为 3001 的元组被组织在一起,3002 的元组被组织在一起,依次类推。然后分别求出各个学生的平均值并输出。

2. HAVING 子句

假如元组在分组前需按照某种方式加以限制,使不需要的分组为空,可以在 GROUP BY 子句后面加一个 HAVING 子句。

注意,当元组含有空值时,应该记住以下两点:

第一,空值在任何聚集操作中都会被忽视,对求和、求平均值和计数都没有影响。它也不能是某列的最大值或最小值。例如,COUNT(*)是某个关系中所有元组数目之和,但 COUNT(A)却是 A 属性非空的元组个数之和。

第二,NULL 值可以看作分组属性中的一个一般的值。例如,在 SELECT A,AVG(B) FORM R 中,当 A 的属性值为空时,就会统计 A=NULL 的所有元组中 B 的均值。

【例 9.15】 针对供应商数据库中的 S、P、J。SPJ 关系,查询哪一个工程至少用了 3 家供应商(包含 3 家)供应的零件的平均数量,并按工程号降序排列。

```
SELECT JNO,AVG(QTY)
FROM SPJ
GROUP BY JNO
HAVING COUNT(DISTINCT(SNO))>2
ORDER BY JNO DESC;
```

根据题意“哪一个工程至少用了 3 家供应商(包含 3 家)供应的零件”,应该按照工程号分组,而且应该加上供应商数目的条件。但需要注意的是,一个工程项目可能用了同一个供应商的多种零件。因此,在统计供应商数的时候需要加上 DISTINCT,以避免重复统计导致错误的结果。

假如按工程号 JNO='J1' 分组,结果如表 9-4 所示。

表 9-4 按工程号 JNO='J1' 分组

Sno	Pno	Jno	Qty
S1	P1	J1	200
S2	P3	J1	400
S2	P3	J1	200
S2	P5	J1	100
S3	P1	J1	200
S4	P6	J1	300
S5	P3	J1	200

从表 9-4 中可以看出,如果不加 DISTINCT,统计的数为 7,而加了 DISTINCT,统计的数是 5。

9.4.6 更名运算

SQL 提供了可为关系和属性重新命名的机制,这是通过如下形式的 AS 子句实现的:

old-name AS new-name

As 子句既可出现在 SELECT 子句中,也可出现在 FROM 子句中。

【例 9.16】 查询计算机学生的 Sname 和 Sage,但 Sname 用姓名表示,Sage 用年龄表示。SQL 语句如下:

```
SELECT Sname AS 姓名,Sage AS 年龄
FROM Students
WHERE Sage<
(SELECT MAX (Sage)
```



```

FROM Students
WHERE SD='CS' )
AND SD<>'CS'

```

SQL 中的元组变量必须和特定的关系相联系。元组变量是通过 FROM 子句中使用 AS 子句来定义的。我们通过例 9.17 来说明。

【例 9.17】 查询计算机选修了 C1 课程的学生姓名 Sname 和成绩 Grade。SQL 语句如下：

```

SELECT Sname,Grade
FROM Students AS x,SC AS y
WHERE x.sno=y.sno and y.cno= 'C1'

```

元组变量在比较同一关系的两个元组是非常有用的。

【例 9.18】 查询平均成绩至少比“1004”平均成绩高的学生学号 Sno 和平均成绩 Grade。其语句如下：

```

SELECT Sno,avg(Grade)
FROM SC as x
Group by sno
Having avg(Grade) > (SELECT avg(Grade)
FROM SC as y
WHERE y.sno= '1004')

```

9.4.7 字符串操作

对于字符串进行的最通常的操作是使用操作符 like 的模式匹配。使用两个特殊的字符来描述模式：“%”匹配任意字符串；“_”匹配任意一个字符。模式是大小写敏感的。例如：

“Marry%”匹配任何以“Marry”开头的字符串；“%idge%”匹配任何包含“idge”的字符串，例如“Marryidge”、“Rock Ridge”、“Mianus Bridge”和“Ridgeway”。

“__”匹配只含两个字符的字符串；“__%”匹配至少包含两个字符的字符串。

【例 9.19】 学生关系模式为 (Sno, Sname, Sex, SD, SAge, SAdd)，其中：Sno 为学号，Sname 为姓名，Sex 为性别，SD 为所在系，SAge 为年龄，SAdd 为家庭住址。请查询：

- (1) 查询家庭住址包含“科技路”的学生姓名。
- (2) 检索名字为“晓军”的学生姓名、年龄和所在系。

解：(1) 家庭住址包含“科技路”的学生姓名的 SQL 语句如下：

```
SELECT Sname
FROM Students
WHERE Add like '%科技路%'
```

(2) 名字为“晓军”的学生姓名、年龄和所在系的 SQL 语句如下:

```
SELECT Sname, Age, SD
FROM Students
WHERE Sname LIKE '__晓军'
```

为了使模式中包含特殊模式字符(如%和_),在 SQL 中允许使用 escape 关键词来定义转义符。转义字符紧靠着特殊字符,并放在它的前面,表示该特殊字符被当成普通字符。例如在 like 比较中使用 escape 关键词来定义转义符,例如使用反斜杠“\”作为转义符。

Like 'ab\%cd%' escape '\', 匹配所有以 ab%cd 开头的字符串。

Like 'ab\\cd%' escape '\', 匹配所有以 ab\cd 开头的字符串。

9.4.8 集合操作

在关系代数中可以用集合的并、交和差来组合关系。SQL 也提供了对应的操作,但是查询的结果必须具有相同的属性和类型列表。保留字 UNION、INTERSECT 和 EXCEPT 分别对应 \cup 、 \cap 和 $-$ 。保留字用于两个查询时,应该分别用括号括起来。

【例 9.20】 假定学生和教师关系模式如下所示,查询既是女研究生,又是教师且工资大于等于 1500 元的名字和地址。

```
Students(Name, Sno, SEX, SD, Type, Address)
Teachers(Name, Eno, SEX, Salary, Address)
```

解: 本题第一条 SELECT 语句和第二条 SELECT 语句查询的结果集模式都为 (Name, Address), 故可以对它们取交集。

```
(SELECT Name, Address
FROM Students
WHERE SEX='女' AND Type='研究生')
INTERSECT
(SELECT Name, Address
FROM Teachers
WHERE Salary >= 1500)
```

同理,我们也可以对两个相同结果集的关系取差集。

【例 9.21】 查询不是教师的学生。

```
(SELECT Name,Address FROM Students)
EXCEPT
(SELECT Name,Address FROM Teachers)
```

9.4.9 视图的查询和删除

1. 视图查询

一旦定义了视图,用户可以象查询基本表那样对视图进行查询。

【例 9.22】 建立“计算机系”(CS 表示计算机系)学生的视图如下,并要求去执行修改、插入操作时保证该视图只有计算机系的学生。

```
CREATE VIEW CS-STUDENT
AS SELECT Sno,Sname,Sage,Sex
FROM Student
WHERE SD='CS'
WITH CHECK OPTION;
```

查询计算机系年龄小于 20 岁的学号及年龄的 SQL 语句如下:

```
SELECT Sno,Sage FORM CS-STUDENT WHERE SD='CS' AND Sage<20;
```

系统执行该语句时,通常先将其转换成等价的对基本表的查询,然后执行查询语句。即当查询视图表时,系统先从数据字典中取出该视图的定义,然后将定义中的查询语句和对该视图的查询语句结合起来,形成一个修正的查询语句。针对上例,修正之后的查询语句为:

```
SELECT Sno,Sage FORM Student WHERE SD='CS' AND Sage<20;
```

2. 视图删除

视图删除的语句如下: DROP VIEW 视图名

3. 视图更新

视图更新必须遵循以下规则:

- (1) 从多个基本表通过连接操作导出的视图不允许更新。
- (2) 对使用了分组、集函数操作的视图不允许进行更新操作。
- (3) 如果视图是从单个基本表通过投影、选取操作导出的,则允许进行更新操作,且语法同基本表。

4. WITH 子句

如果我们将一个复杂的查询分解成一些小视图,然后将它们组合起来,就像将一个程

序按其任务分解成一些过程一样,使得复杂查询的编写和理解都会简单得多。然而和过程定义不同的是 CREATE VIEW 子句会在数据库中建立视图定义,该视图定义会一直保存,直到执行 DROP VIEW 命令。但是,WITH 子句提供了定义一个临时视图的方法,该定义只对随 WITH 子句出现的查询有效。

【例 9.23】 假定教师关系模式为 Teachers (TName, Eno, Tdept, SEX, Salary, Address),利用 WITH 子句查询工资最高的教师姓名。此时,如果具有同样工资最高的教师有多个,他们都会被选择。

解:

```
WITH max-Salary(value) AS
  SELECT max(Salary)
    FROM Teachers
  SELECT Tname
    FROM Teachers,max-Salary
  WHERE Teachers. Salary = max-Salary. value
```

WITH 子句是在 SQL-99 中引入的,目前只有部分数据库支持这一子句。

用 FROM 子句或 WHERE 子句中的嵌套子查询也能实现上述查询,但是采用这样的方法会使查询语句可读性差。WITH 子句使查询逻辑上更加清晰,它还允许在一个查询中多处使用同一个数据定义。

【例 9.24】 假定银行账户关系模式为 Account(Account-no,branch-name,balance),其中属性 Account-no 表示账号,branch-name 表示支行名称,balance 表示余额。利用 WITH 子句查询所有存款总额少于所有支行平均存款总额的支行。

解:

```
WITH branch-total(branch-name,value) AS
  SELECT branch-name,sum(balance)
    FROM Account
  GROUP BY branch-name
WITH branch-total-avg(value) AS
  SELECT avg(value)
    FROM branch-total
SELECT branch-name
  FROM branch-total,branch-total-avg
 WHERE branch-total. value >= branch-total-avg. value
```


9.4.10 插入、删除和修改语句

1. 插入语句

在关系数据库中插入数据,可以指定被插入的元组,或者用查询语句选出一批待插入的元组。插入语句的基本格式如下:

```
INSERT INTO 基本表名(字段名[,字段名]...)
VALUES(常量[,常量]...); 查询语句
INSERT INTO 基本表名(列表名)
SELECT 查询语句
```

【例 9.25】 将学号为“3002”、课程号为“C4”以及成绩为 98 的元组插入 SC 关系中。其语句如下:

```
Insert into SC
Values('3002', 'C4', 98)
```

【例 9.26】 首先创建一个新的视图 v_employees(该视图是基于表 employees 创建的),

```
create view v_employees(number, name, age, sex, salary)
as
select number, name, age, sex, salary
from employees
where name='张三'
```

然后通过执行以下语句使用该视图向表 employees 中添加一条新的数据记录。

```
Insert into v_employees
Values(001, '李力', 22, 'm', 2000)
```

2. 删除语句

删除语句的语法为: DELETE FROM 基本表名
[WHERE 条件表达式]

【例 9.27】 从表 employees 中删除姓名为张然的记录。

```
DELETE from employees
where name='张然'
```

3. 修改语句

修改语句的表达式为:

UPDATE 基本表名

SET 列名=值表达式(,列名=值表达式...)

[WHERE 条件表达式]

【例 9.28】 将教师的工资增加 5%。

```
UPDATE teachers
```

```
set Salary = Salary * 1.05
```

【例 9.29】 将工资小于 1000 的教师增加 5%工资。

```
UPDATE teachers
```

```
SET Salary = Salary * 1.05
```

```
WHERE Salary <= 1000
```

使用视图可以更新数据记录,但应该注意的是,更新的只是数据库中的基表。

【例 9.30】 创建一个基于表 employees 的视图 v_employees,然后通过该视图修改表 employees 中的记录。其语句如下:

```
CREATE view v_employees
```

```
AS
```

```
SELECT * from employees
```

```
UPDATE v_employees
```

```
SET name='张然'
```

```
WHERE name='张三'
```

9.5 SQL 中的授权

数据库的完整性是指数据库的正确性和相容性,是防止合法用户使用数据库时向数据库加入不符合语义的数据,保证数据库中数据是正确的,避免非法的更新。数据库完整性需要掌握的重点内容有:完整性约束条件的分类,完整性控制应具备的功能。

9.5.1 主键约束 PRIMARY KEY

1. 完整性约束条件

完整性约束条件作用的对象有关系、元组和列 3 种,共分为 6 类,见表 9-5。

2. 完整性控制

完整性控制应具有三方面的功能:定义功能、检测功能和处理功能(一旦发现违背了完整性约束条件,采取相关的措施来保证数据的完整性)。

表 9-5 完整性约束条件

状态	约束对象	说 明
静态	列级约束	是对一个列的取值域的说明。包括：数据类型的约束、数据格式的约束(如 YY, MM, DD)、取值范围的约束以及空值的约束
	元组约束	规定元组各列之间的约束关系。如教师关系中的职称和工资,规定教授的工资不得低于 1000 元
	关系约束	实体完整性约束、参照完整性约束、函数依赖约束以及统计约束
动态	列级约束	修改列定义时的约束、修改列值时的约束
	元组约束	指修改元组值时元组中各个字段间需要满足的某种约束条件。如工资 = 基本工资 + 工龄 * 2
	关系约束	是加在关系变化前后状态上的限制条件

检查是否违背完整性约束的时机有两种：若在一条语句执行完后立即检查称为立即执行约束；若需要延迟到整个事务执行完后再检查称为延迟执行约束。

数据库中最重要约束是声明一个或一组属性形成关系的键。键的约束在 SQL 的 CREATE TABLE 命令中声明。在关系系统中,最重要的完整性约束条件是实体完整性和参照完整性。

3. 实体完整性(使用“PRIMARY KEY”子句)

关系中只能有一个主键。声明主键有两种方法：

- (1) 将 PRIMARY KEY 保留字加在属性类型之后；
- (2) 在属性列表中引入一个新元素,该元素包含保留字 PRIMARY KEY 和用圆括号括起的构成键的属性或属性组列表。

【例 9.31】 针对学生关系 Students(Sno, Sname, Sex, Sdept, Sage),可使用如下语句创建表：

```
CREATE TABLE Students
( Sno CHAR(8),
  Sname CHAR(10),
  Sex CHAR(1),
  Sdept CHAR(20),
  Sage NUMBER(3),
  PRIMARY KEY(Sno));
```

或采用如下方法：

```
CREATE TABLE Students
( Sno CHAR(8) PRIMARY KEY,
```

```
Sname CHAR(10),  
Sex CHAR(1),  
Sdept CHAR(20),  
Sage NUMBER(3));
```

当主键有多个属性时必须用第一种方法。

【例 9.32】 针对学生选课关系 SC(Sno,Cno,Grade),可使用如下语句创建表:

```
CREATE TABLE SC  
( Sno CHAR(8),  
  Cno CHAR(4),  
  Grade NUMBER(3),  
  PRIMARY KEY(Sno),  
  PRIMARY KEY(Cno));
```

9.5.2 外键约束 FOREIGN KEY

参照完整性定义格式如下:

```
FOREIGN KEY(属性名)REFERENCES 表名(属性名)  
[ON DELETE[CASCADE|SET NULL]]
```

参照完整性通过保留字 FOREIGN KEY 定义哪些列为外码,REFERENCES 指明外码对应于哪个表的主码,ON DELETE CASCADE 指明删除参照关系的元组时,同时删除参照关系中的元组,SET NULL 表示置为空值方式。

【例 9.33】 在例 9.32 学生选课关系 SC(Sno,Cno,Grade)中,学号 Sno 参照关系 Student,课程号 Cno 参照关系 C。因此对于例 9.32,正确的语句为:

```
CREATE TABLE SC  
( Sno CHAR(8),  
  Cno CHAR(4),  
  Grade NUMBER(3),  
  PRIMARY KEY(Sno),  
  PRIMARY KEY(Cno),  
  FOREIGN KEY Sno REFERENCES Students(Sno),  
  FOREIGN KEY Cno REFERENCES C(Cno));
```

9.5.3 属性值上的约束

1. NULL

【例 9.34】 设有学生关系 Students(Sno,Sname,Sex,Sdept,Sage),如果要求学生姓

名不能为空,那么可使用如下语句创建表:

```
CREATE TABLE Students
( Sno CHAR(8),
  Sname CHAR(10) NOT Null,
  Sex CHAR(1),
  Sdept CHAR(20),
  Sage NUMBER(3),
  PRIMARY KEY(Sno));
```

2. CHECK

【例 9.35】 在 Students 中,要求男生的年龄在 15~25 岁之间,女生的年龄在 15~24 岁之间。

解:根据题意,在关系 Students 的定义中增加一条检查子句(基于元组的检查子句举例):

```
CREATE TABLE Students
( Sno CHAR(8),
  Sname CHAR(10),
  Sex CHAR(1),
  Sdept CHAR(20),
  Sage NUMBER(3),
  PRIMARY KEY(Sno))
CHECK (Sage >=15 AND ((SEX='M' AND Sage <=25)OR
                      (SEX='F' AND Sage <25)));
```

9.5.4 全局约束 CREATE ASSERTIONS

全局约束是指一些比较复杂的完整性约束,这些约束涉及到多个属性间的联系或多个不同关系间的联系。全局约束有两种:基于元组的检查子句和断言。

(1) 基于元组的检查子句

这种约束是对单个关系的元组值加以约束。方法是在关系定义中的任何地方加上关键字 CHECK 和约束条件。

例如,年龄在 16 至 20 岁之间,可用 CHECK (Sage>=16 AND Sage<=20)检测。

(2) 基于断言的语法格式

格式: CREATE ASSERTION <断言名> CHECK(<条件>)

【例 9.36】 在教学数据库模式 Students、SC 和 C 中加一个约束,不允许男同学选修

“张勇”老师的课。

解：根据题意可写成如下的断言形式(基于断言的举例)：

```
CREATE ASSERTION ASSE-SC1 CHECK
  (NOT EXISTS
    (SELECT * FROM SC WHERE Cno IN
      (SELECT Cno FROM C WHERE TEACHER='张勇')
      AND Sno IN
      (SELECT Sno FROM Students WHERE SEX='M')));
```

【例 9.37】 教学数据库模式 Students、SC 和 C 中有一个约束，每门课最多允许 50 名男同学选修。

解：根据题意可写成如下的断言形式：

```
CREATE ASSERTION ASSE-SC2 CHECK
  (50 >= ALL(SELECT COUNT(SC. Sno)
    FROM Students, SC
    WHERE Students. Sno = SC. Sno AND SEX = 'M'
    GROUP BY Cno));
```

9.5.5 授权与销权

数据控制的目的是控制用户对数据的存储权力，是由 DBA 来决定的。但是，某个用户对某类数据具有何种权利，是个政策问题而不是技术问题。DBMS 的功能就是保证这些决定的执行。因此，DBMS 数据控制应具有如下功能：

- 通过授权(GRANT)和销权(REVOKE)将授权通知系统，并存入数据字典。
- 当用户提出请求时，根据授权情况检查是否能够执行操作请求。

1. 授权语句格式

```
GRANT <权限>[, <权限>]...
  [ON<对象类型><对象名>]
  TO <用户>[, <用户>]>...
  [WITH GRANT OPTION];
```

注意：不同类型的操作对象有不同的操作权限，常见的操作权限见表 9-6。

说明：

- PUBLIC：接受权限的用户可以是单个或多个具体的用户，PUBLIC 参数可将权限赋给全体用户。

表 9-6 常见的操作权限

对象	对象类型	操 作 权 限
属性列	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES(5 种权限的总和)
视图	TABLE	SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES(5 种权限的总和)
基本表	TABLE	SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX ALL PRIVILEGES(6 种权限的总和)
数据库	DATABASE	CREATETAB 建立表的权限,可由 DBA 授予普通用户

- WITH GRANT OPTION: 若指定了此子句,那么,获得权限的用户还可以将权限赋给其他用户。

【例 9.38】 将对供应商 S、零件 P 以及项目 J 的所有操作权限赋给用户 User1 及 User2。

GRANT ALL PRIVILEGES ON TABLE S,P,J TO User1, User2;

【例 9.39】 将对供应商 S 的插入权限赋给用户 User1,并允许将此权限赋给其他用户。

GRANT INSERT ON TABLE S TO User1 WITH GRANT OPTION;

【例 9.40】 DBA 把数据库 SPJ 中建立表的权限赋给用户 User1。

GRANT CREATETAB ON DATABASE SPJ TO User1;

2. 收回权限语句格式

REVOKE <权限>[,<权限>]...
 [ON<对象类型><对象名>]
 FROM <用户>[,<用户>]...;

【例 9.41】 将用户 User1 及 User2 对供应商 S、零件 P 和项目 J 的所有操作权限收回。

REVOKE ALL PRIVILEGES ON TABLE S,P,J FROM User1, User2;

【例 9.42】 将所有用户对供应商 S 的所有查询权限收回。

REVOKE SELECT ON TABLE S FROM PUBLIC;

【例 9.43】 将 User1 用户对供应商 S 的供应商编号 Sno 的修改权限收回。

REVOKE UPDATE(Sno) ON TABLE S FROM User1;

9.6 触发器

9.6.1 概述

触发器是一种特殊类型的存储过程,与前面介绍过的存储过程不同,它是通过事件触发而执行的,而存储过程可以通过存储过程名称被直接调用。触发器使每个站点在有数据修改时自动强制执行其业务规则,并且可以用于 SQL Server 约束、默认值和规则的完整性检查。

触发器的主要特点如下:

- 当数据库程序员声明的事件发生时,触发器被激活。事件可以是对某个特定关系的插入、删除或更新。
- 当触发器被事件激活时,不是立即执行,而是首先由触发器测试触发条件,若条件不成立,响应该事件的触发器什么事情都不做。
- 如果触发器声明的条件满足,则由 DBMS 执行与该触发器相连的动作。该动作可以阻止事件发生,可以撤销事件。

需要注意的是,触发器为数据库对象。当创建一个触发器时,必须指定:①名称;②在其上定义触发器的表;③触发器将何时激发;④触发器执行时应做的动作。其名称必须遵循标识符的命名规则,数据库像存储普通数据那样存储触发器。触发器可以引用当前数据库以外的对象,但只能在当前数据库中创建触发器。尽管不能在临时表或系统表上创建触发器,但是触发器可以引用临时表。

9.6.2 创建触发器

对于示警或满足特定条件下自动执行某项任务来讲,触发器是非常有用的机制。假设银行在处理透支时,不是将账户余额设置成负值,而是将账户余额设置成零,并且建立一笔贷款,其金额为透支额。这笔贷款的贷款号等于该透支账户的账户号。执行触发器的条件是对关系 Account 的更新导致了负的余额。

虽然在 SQL-99 之前,触发器并不是 SQL 标准的一部分,以 SQL 为基础的数据库还是广泛应用了触发器。但是,不同的数据库系统使用的是各自的触发器语法,导致相互不兼容。

【例 9.44】 假定银行数据库关系模式如下:

Account (Account-no, branch-name, balance) Loan(Loan-no,branch-name,amount) depositor(customer-name,Account-no)

账户关系模式 Account 中的属性 Account-no 表示账号, branch-name 表示支行名称, balance 表示余额。贷款关系模式 Loan 中的属性 Loan-no 表示贷款号, branch-name 表示支行名称, amount 表示金额。存款关系模式 depositor 中的属性 customer-name 表示存款人姓名。采用 SQL-99 标准创建触发器如下:

```
CREATE TRIGGER overdraft_trigger after update on Account
Referencing new row as nrow
For each row
When nrow. balance<0
Begin atomic
    Insert into borrower
        (SELECT customer-name, Account-no
         FROM depositor
         Where nrow. account-no=depositor. account-no);
    Insert into values
        (nrow. account-no,nrow. branch-name,-nrow. balance);
    update account set balance=0
        Where account. account-no=nrow. account-no
End
```

When 语句指定了条件 `nrow. balance<0`, 表示仅对满足条件的元组才会执行下面的触发器。

Begin atomic...End 子句用来将多行 SQL 语句集成为一个复合语句。该子句中的两条 Insert into 语句表示在 borrower 和 loan 关系中建立新的贷款业务, update 语句用来将账户余额清零。

Referencing old row as 子句可以建立一个变量, 用来存储已经被更新或删除的行的旧值。Referencing new row as 子句可以被 update 和 Insert 语句使用。

Referencing old table as 或 Referencing new table as 子句可以用来指向临时表(也称过渡表), 使之容纳所有被影响的行。无论临时表是语句触发器还是行触发器, 它们都不能用 before 触发器, 但可以用 after 触发器。

触发器事件和动作可以有很多形式, 除了 update 以外还可以是 Insert 和 delete。例如, 如果有一个新的用户存款执行插入操作时, 触发器可以给用户发一封欢迎信。显然,

一个触发器不能直接对数据库之外的事情进行操作,但它可以在存储待发欢迎信地址的关系中添加一个元组。一个专用进程会检查这个表,打印出要发出的欢迎信。

触发器在事件(update、Insert 和 delete)之前被激发,而不是在事件之后。这种触发器可作为避免非法更新的额外约束。例如,不允许用户透支时,可以建立一个 before 触发器在新的余额是负数时回滚事务。

【例 9.45】 设仓库管理数据库中有如下关系:

inventory(item,level),表示仓库中某种商品的现有量。

minlevel(item,level),表示仓库中存有某种商品的最小量。

reorder(item,amount),表示某种商品小于最小量时要订购的数量。

orders(item,amount),表示订购某种商品的量。

下面是一个重新订购商品的触发器:

```
CREATE TRIGGER reorder_trigger after update of amount on inventory
Referencing old row as orow,new row as nrow
For each row
When nrow.level <= (SELECT level
                     FROM minlevel
                     Where minlevel.item = orow.item)
And orow.level > (SELECT level
                  FROM minlevel
                  Where minlevel.item = orow.item)
Begin
    Insert into orders
        (SELECT item,amount
         FROM reorder
         Where reorder.item = orow.item)
End
```

9.6.3 删除触发器

使用系统命令 DROP TRIGGER 删除指定的触发器,其语法格式如下:

```
DROP TRIGGER {trigger} [ ... n ]
```

(1) 删除触发器所在的表时,SQL Server 将会自动删除与该表相关的触发器。

(2) 在企业管理器中,用右键单击要删除的触发器所在的表,从弹出的快捷菜单中选择所有任务子菜单下的管理触发器选项,则会出现触发器属性对话框。在名称选项框中选择要删除的触发器,单击“删除”按钮,即可删除该触发器。

9.7 嵌入式 SQL

9.7.1 SQL 与宿主语言接口

SQL 提供了将 SQL 语句嵌入某种高级语言中的使用方式。但如何识别嵌入在高级语言中的 SQL 语句呢？通常采用预编译的方法。该方法的关键问题是必须区分主语言中嵌入的 SQL 语句，以及主语言和 SQL 间的通信问题。通常是由 DBMS 预处理程序对源程序进行扫描，识别出 SQL 语句。然后把它们转换为主语言调用语句，使主语言的编译程序能识别它。最后由主语言的编译程序将整个源程序编译成目标码。

可见，将 SQL 嵌入主语言时应当注意如下问题。

1. 区分主语言语句与 SQL 语句

为了区分主语言语句与 SQL 语句，需要在所有的 SQL 语句前加上前缀 EXEC SQL，而 SQL 的结束标志随主语言的不同而不同。

例如，PL/1 和 C 语言的引用格式为：

```
EXEC SQL <SQL 语句>;
```

又如，COBOL 语言的引用格式为：

```
EXEC SQL <SQL 语句> END-EXEC
```

2. 主语言工作单元与数据库工作单元通信

1) SQL 通信区

SQL 通信区 (SQL Communication Area, SQLCA) 向主语言传递 SQL 语句执行的状态信息，使主语言能够根据此信息控制程序流程。

2) 主变量

主变量也称共享变量。主语言主要通过主变量向 SQL 语句提供参数。主变量由主语言的程序定义，并用 SQL 的 DECLARE 语句说明。例如，在 C 语言中可用如下形式说明主变量：

```
EXEC SQL BEGIN DECLARE SECTION;           /* 说明主变量 */
    char Msno[4], Mcno[3], givensno[5];
    int Mgrade;
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

上面 5 行组成一个说明节，说明了 5 个共享变量。其中，SQLSTATE 是一个特殊的

共享变量,起着解释 SQL 语句执行状况的作用。当 SQL 语句执行成功时,系统自动为 SQLSTATE 赋全零值,否则为非全零值(“02000”)。因此,当执行一条 SQL 语句后,可以根据 SQLSTATE 的值转向不同的分支,以控制程序的流向。引用时,为了与 SQL 属性名相区别,需在主变量前加“:”。

【例 9.46】 根据共享变量 givensno 值查询学生关系 students 中学生的姓名、年龄和性别。

```
EXEC SQL SELECT sname,age,sex
        INTO :Msno,:Mcno,:givensno
        FROM students
        WHERE sno=:Msno;
```

3) 游标

SQL 语言是面向集合的,一条 SQL 语句可产生或处理多条记录。而主语言是面向记录的,一组主变量一次只能存一条记录。所以,引入了游标概念。通过移动游标指针可以决定获取哪一条记录。与游标相关的 SQL 语句有以下四条。

(1) 定义游标。

定义游标的格式如下:

```
EXEC SQL DECLARE <游标名> CURSOR FOR
        <SELECT 语句>
END_EXEC
```

这是一条说明性语句,定义中的 SELECT 语句并不立即执行。

(2) 打开游标。

打开游标的格式如下:

```
EXEC SQL OPEN <游标名> END_EXEC
```

该语句执行游标定义中的 SELECT 语句,同时使游标处于活动状态。游标是一个指针,此时指向查询结果的第一行之前。

(3) 推进游标。

推进游标的格式如下:

```
EXEC SQL FETCH FROM <游标名> INTO <变量表> END_EXEC
```

执行该语句时,游标推进一行,并把游标所指行(称为当前行)中的值取出,送到共享变量中。变量表由使用逗号分开的共享变量组成。该语句常置于宿主语言程序的循环结构中,并借助宿主语言的处理语句逐一处理查询结果中的一个元组。

(4) 关闭游标。

关闭游标的格式如下：

```
EXEC SQL CLOSE <游标名> END_EXEC
```

该语句关闭游标,使它不再和查询结果相联系。关闭了的游标可以再次打开,使之与新的查询结果相联系。在游标处于活动状态时,可以修改和删除游标指向的元组。

【例 9.47】 在 C 语言中嵌入 SQL 查询,检索某学生的学习成绩,其学号由共享主变量 givensno 给出,结果放在主变量 Sno,Cno 和 Grade 中。如果成绩不及格,则删除该记录。如果成绩在 60~69 分之间,则将成绩修改为 70 分,并显示学生的成绩信息(不含 60 分以下的)。

解：

```
# DEFINE NO_MORE_TUPLES ! (strcmp(SQLSTATE,"02000"))
void sel()
{ EXEC SQL BEGIN DECLARE SECTION;          /* 说明主变量 */
  char Msno[4], Mcno[3], givensno[5];
  int Mgrade;
  char SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;
  EXEC SQL DECLARE Scx CURSOR FOR          /* 说明游标 Scx,将查询结 */
    SELECT Sno,Cno,Grade                  /* 果与 Scx 建立联系 */
    FROM SC
    WHERE Sno=,givensno;
  EXEC SQL OPEN Scx;
  While(1)                                /* 用循环结构逐条处理结果集中的记录 */
  { EXEC SQL FETCH FROM Scx                /* 游标推进一行 */
    INTO ,Msno,,Mcno,,Mgrade;             /* 送入主变量 */
    If (NO_MORE_TUPLES) Break;             /* 处理完退出循环 */
    If (Mgrade<60)                         /* 成绩<60 */
      EXEC SQL DELETE FROM Sc
      WHERE CURRENT OF Scx;
    Else
      { If (Mgrade<70)                    /* 成绩<70 */
        { EXEC SQL UPDATE Sc
          SET grade=70
          WHERE CURRENT OF Scx;
          MGrade=70
        }
        Printf("%s,%s,%d",Msno,Mcno,Mgrade); /* 显示学生记录 */
      }
  }
};
```

9.7.2 动态 SQL

SQL 的动态组件允许程序构造和提交 SQL 查询。与此相反,嵌入式 SQL 语句必须在编译时完全确定,由预处理程序预编译和宿主语言编译程序编译。也就是说,在实际使用时,源程序往往不能包括用户的所有操作。用户对数据库的操作有时往往在实际运行时才提出来,为此需要采用动态 SQL 技术。动态 SQL 有以下两条语句。

1) 动态 SQL 预备语句格式

EXEC SQL PREPARE <动态 SQL 语句名> FROM <共享变量或字符串>;

此处的共享变量或字符串应该是一个完整的 SQL 语句。这个 SQL 语句可以在程序运行时根据用户输入才组合起来,但并不执行。

2) 动态 SQL 执行语句格式

EXEC SQL EXECUTE <动态 SQL 语句名>;

使用动态 SQL 语句时,还可以有两点改进技术:

当由预备语句组合而成的 SQL 语句只需执行一次时,预备语句可以在程序运行时根据用户的输入再组合起来,但并不执行。

9.8 SQL-99 所支持的对象关系模型

对象-关系数据模型扩展关系数据模型的方式是通过增加复杂的数据类型和面向对象的更丰富的类型系统实现的。关系查询语言(特别是在 SQL 中)也需要做相应的扩展以处理这些更丰富的类型系统。这种扩展试图在扩展建模能力的同时保留关系的基础——特别是对数据的说明性存取。对象-关系数据库系统是以对象-关系模型为基础的数据库系统。

9.8.1 嵌套关系

前面定义的第一范式(1NF)指关系模式中的属性是不可再分的,即原子的。然而,并不是所有的应用都用 1NF 建模才是最好的。例如,某些应用的用户将数据库视为对象(或实体)的一个集合,而不是记录的一个集合,这些对象可能需要用若干条记录表示。重要的是,如何在一个简单易用的界面上,体现用户直观概念上的一个对象与数据库系统概念上的一个数据项之间的一一对应关系。

嵌套关系模型(nested relational model)是关系模型的一个扩展,域可以是原子的也可以赋值为关系。这样元组在一个属性上的取值可以是一个关系,于是关系可以存储在关系中。从而一个复杂对象就可以用嵌套关系的单个元组来表示。如果将嵌套关系的一

个元组视为一个数据项,在数据项和用户数据库观念上的对象之间就有了一个一一对应的关系。

下面通过一个图书馆的例子来说明嵌套关系。假定对每本书都存储如下信息:

书名	作者集合	出版商	关键字集合
----	------	-----	-------

如果为上述信息定义一个关系,下列一些域将是非原子的:

- 作者。一本书可能有一组作者,然而,我们可能想要寻找作者之一是 Jones 的所有书,这样我们就对域元素“作者集合”的一个子部分感兴趣。
- 关键字。如果为一本书存储了一组关键字,希望能够检索出包含该集合中的一个或多个关键字的所有书,这样就将关键字集合域视为非原子的。
- 出版商。与关键字和作者不同,出版商没有一个以集合为值的域。但是,可能将出版商视为由名字和分支机构这两个子字段组成的,这种观念使得出版商域成为非原子的。

图 9-2 给出了一个示例关系 books。books 关系可以用 1NF 表示,如图 9-3 所示。由于 1NF 中只能有原子的域,而又想要访问单个的作者和单个的关键字,每个(关键字,作者)对都需要一个元组。出版商属性在 1NF 版本中被两个属性取代,它们分别对应于出版商的每个子字段。

Title	Author-set	publishér	keyword-set
		(name, branch)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

图 9-2 非 1NF 的书籍关系 books

title	author	pub-name	pub-branch	keyword
Compilers	Smith	McGraw-Hill	New York	Parsing
Compilers	Jones	WcGraw-Hill	New York	Parsing
Compilers	Smith	McGraw-Hill	New York	Analysis
Compilers	Jones	McGraw-Hill	New York	Analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

图 9-3 flat-books 的一个 1NF 实例

如果假定下列多值依赖成立：

- title \twoheadrightarrow author
- title \twoheadrightarrow keyword
- title \twoheadrightarrow pub-name, pub-branch

可以使用下面的模式将这个关系分解成 4NF, 就可以去除图 9-3 中 flat-books 关系的冗余。图 9-4 显示了图 9-3 中的 flat-books 关系到上述分解的映射。

- authors(title, author)
- keywords(title, keyword)
- books4(title, pub-name, pub-branch)

title	author	title	Keyword
Compilers	Smith	Compilers	parsing
Compilers	Jones	Compilers	analysis
Networks	Jones	Networks	Internet
Networks	Frick	Networks	Web

authors		keywords	
Title	pub-name	pub-branch	
Compilers	McGraw-Hill	New York	
Networks	Oxford	London	

books4

图 9-4 关系 flat-books 的 4NF 实例

尽管不用嵌套关系也足以表达上述示例书籍数据库, 但嵌套关系可以产生一个更易理解的模型: 一个信息检索系统的典型用户根据具有作者集的书将该数据库看成一个非 1NF 的设计模型。4NF 设计会要求用户在他们的查询中包含连接操作, 因此使得与系统的交互复杂化。

我们可以设计一个无嵌套的关系视图(它的内容与 flat-books 一样), 以免除用户在他们的查询中增加连接操作。然而在这样的视图中, 我们却失去了元组与书之间的一对一的对应。

面向对象数据库系统支持面向对象数据模型, 是一个持久的、可共享的对象库的存储和管理者, 而一个对象库是由一个 OO 模型所定义的对象集合体。

对象: 是由一组数据结构和在这组数据结构上的操作的程序代码封装起来的基本单位。对象之间的界面由一组消息定义。一个对象包括属性集合、方法集合和消息集合。

对象标识: 是指面向对象数据库中的每个对象都有一个惟一不变的标识。常用的几种标识有值标识、名标识和内标识。

- 值标识,使用一个值来标识,关系数据库中通常使用这种形式的标识。例如,一个元组的主码标识了这个元组。
- 名标识,用用户提供的名称作为标识。这种形式的标识通常用于文件系统中的文件,不管文件的内容是什么,每个文件都被赋予一个名称作为惟一标识。
- 内标识,是建立在数据模型或程序设计语言中内置的一种标识,不需要用户给出标识。面向对象系统中使用这种形式的标识,每个对象在创建时都被系统自动赋予一个标识符。

封装: OO 模型的一个关键概念就是封装。每一个对象都是其状态和行为的封装。封装是对对象的外部界面与内部实现之间实行清晰隔离的一种抽象,外部与对象的通信只能通过消息。

类: 共享同样属性和方法集的所有对象构成了一个对象类(简称类)。例如,学生是一个类,黎明、张军和樊建喜是学生类中的一个对象。类是“型”,对象是“值”。

9.8.2 复杂类型

嵌套关系只是对基本关系模型扩展的一个实例,其他非原子数据类型,如嵌套记录,同样已被证明是有用的。面向对象数据模型已经导致了对于诸如对象的继承和引用之类特征的需求。有了复杂对象系统和面向对象,能够直接表达 E-R 模型的一些概念,如实体标识、多值属性及一般化和特殊化,而不再需要经过关系模型的复杂转化。

本节通过描述为允许复杂类型对 SQL 所做的扩展,包括嵌套关系以及面向对象特征,来介绍基于 SQL-99 标准的一些特征。

1. 集合类型

考虑下列代码段:

```
create table books(
    ...
    keyword-set setof(varchar(20))
    ...
)
```

这个表的定义不同于普通关系数据库中表的定义,因为它允许属性是集合(set),所以 E-R 图中的多值属性能够直接表示。

集合是集合体类型(collection type)的一个实例,其他集合体类型包括数组(array)和多重集合(multiset)(即无序的集合体,其中一个元素可以出现多次),下面的定义声明了一个数组:

```
author-array varchar(20) array [10]
```

这里的 `author-array` 是一个最多容纳 10 位作者名的数组。我们可以通过指定数组下标来访问数组中的元素,例如 `author-array[1]`。数组是 SQL-99 中惟一支持的集合体类型,使用语法如前所述。SQL-99 不支持无序集合或多重集合,尽管它们可能会在 SQL 将来的版本中出现。

现在的许多数据库应用需要存储的属性很大(大约几千字节),如一个人的相片,或者更大的(大约几兆甚至上千兆字节),如高分辨率的医学图像或者录像剪辑。因此 SQL-99 提供了字符型数据大对象数据类型(clob)和二进制数据大对象数据类型(blob)。数据类型中的“lob”意为“Large Object”。示例如下。

```
book-review clob(10KB)
image blob(10MB)
movie blob(2GB)
```

大对象一般用于外部的应用。通过 SQL 对它们进行全体检索是毫无意义的。相反,应用程序一般只检索大对象的“定位器”,然后用定位器从宿主语言中操作该对象。例如, JDBC 允许程序员分成小片来存取一个大对象,而不是一次全取出来,这很像从操作系统文件中存取数据。

2. 结构类型

在 SQL-99 中的结构类型声明以及使用方法如下例所示。

【例 9.48】 定义一个 `Publisher` 类型,其中包括名字(name)和分支机构(branch)两个部分。定义一个结构类型 `Book`,其中包含标题(title)、作者数组(author-array)、出版时间(pub-date)、出版商(publisher,是 `Publisher` 类型的)和一个关键字集合(keyword-set 作为集合的声明使用了扩展语法,SQL-99 标准中并不支持)。

```
create type Publisher as
  (name varchar(20),
   branch varchar(20))
create type Book as
  (title varchar(20),
   author-array varchar(20) array [10],
   pub-date date,
   publisher Publisher,
   keyword-set setof(varchar(20)))
create tabel books of Book
```

执行上述语句后,可以创建一个包含 `Book` 类型元组的表 `books`,这个表类似于图 9-2

中的嵌套关系 books, 创建一个作者名的数组以代替作者名的集合, 数组允许按顺序记录作者名。E-R 图中的复合属性可以用结构类型直接表达。在 SQL-99 中, 无名称的行类型 (row type) 用来定义复合属性。

【例 9.49】 定义属性 publisher1 为无名称的行类型。

```
publisher1 row(name varchar(20),  
               branch varchar(20))
```

需要说明的是, 例 9.47 中的 books 表创建了一个中间类型 book。事实上, 也可以不创建中间类型, 直接将 books 表定义如下:

```
create type Publisher as  
    (name varchar(20),  
     branch varchar(20))  
create table books  
    (title varchar(20),  
     author-array varchar(20) array[10],  
     pub-date date,  
     publisher Publisher,  
     keyword-set setof(varchar(20)))
```

在上述声明中, 该表的行没有明确的类型。结构类型还可以有定义在其上的方法 (method), 而且可以将方法的声明作为一个结构类型定义的一部分:

```
create type Employee as(  
    name varchar(20),  
    salary integer)  
method giveraise (percent integer)
```

然后单独创建方法的主体:

```
create method giveraise (percent integer) for Employee  
begin  
    set self.salary = self.salary + (self.salary * percent)/100;  
end
```

变量 self 是指调用这个方法的结构类型的实例。方法的主体可以包含过程语句。

3. 复杂类型值的创建

SQL-99 中的构造器函数 (constructor function) 可以用来创建结构类型的值。与结构类型同名的函数就是这个结构类型的构造器函数。

【例 9.50】 为 Publisher 类型声明一个构造器。

```
create function Publisher (n varchar(20), b varchar(20))
returns Publisher
begin
    set name=n;
    set branch=b;
end
```

然后,我们可以用 Publisher('McGraw-Hill', 'New York') 创建 Publisher 类型的值。

SQL-99 也支持除构造器之外的其他函数。这些函数的名字必须不同于任何结构类型的名字。注意,不同于面向对象数据库,SQL-99 中的构造器创建的是类型的一个值,而不是类型的一个对象。也就是说,构造器创建的值没有对象标识。在 SQL-99 中,对象相当于关系中的元组,可通过在关系中插入一个元组来创建对象。

默认的情况下,每一个结构类型都有一个不带参数的构造器,它将属性设为默认值。其他任何构造器必须被显式地创建。同一个结构类型可以有不止一个构造器,虽然它们有一个相同的名字,但它们必须以参数的个数和类型来相互区别。

在 SQL-99 中,可以这样创建数组的值:

```
array['Silberschatz', 'Korth', 'Sudarshan']
```

我们可以通过在圆括号中列出它的属性来构造一行的值。例如,我们可以声明 Publisher1 属性为一个行类型,这样构造它的值:

```
('McGraw-Hill', 'New York')
```

这里没有用到构造器。

可以通过在关键字 set 之后的圆括号内列举元素的方法创建以集合为值的属性,如 keyword-set。可以像集合值一样创建多重集合值,此时只需要用 multiset 替换 set。

可以创建用 books 关系定义的类型的一个元组:

```
('Compilers', array['Smith', 'Jones'], Publisher('McGraw-Hill', 'New York'),
    set('parsing', 'analysis'))
```

利用合适的参数调用 Publisher 的构造器函数,可以创建 Publisher 属性的值。如果想在 books 关系中插入上述元组,可以执行语句:

```
insert into books
values
('Compilers', array['Smith', 'Jones'], Publisher('McGraw-Hill', 'New York'),
    set('parsing', 'analysis'))
```


9.8.3 继承

继承可以在类型的级别上进行,也可以在表级别上进行,下面分别介绍。

1. 类型继承

若希望在数据库中对那些是学生和教师的人分别存储一些额外的信息,需要使用类型继承。

【例 9.51】 假定人的类型定义如下所示,定义学生和教师类型。

```
create type Person
(name varchar(20),
address varchar(20))
```

由于学生和教师是人,所以可以使用继承。按照 SQL-99 标准定义学生和教师类型如下:

```
create type Student
under Person
(degree varchar(20),
department varchar(20))
create type Teacher
under Person
(salary integer,
department varchar(20))
```

Student 和 Teacher 都继承了 Person 的属性,即 name 和 address。Student 和 Teacher 被称为 Person 的子类型,Person 是 Student 的超类型,同时也是 Teacher 的超类型。像属性一样,结构类型的方法也被它的子类型继承。不过,子类型可以通过在一个方法声明中使用 overriding method(重载方法)取代原 method(方法)的方式重新声明方法,以重定义该方法的作用。

现在假定要存储关于助教的信息,这些助教既是学生又是教师,甚至可能是在不同的系里。可以利用多重继承(multiple inheritance)的方法来做。SQL-99 标准原来是准备提供多重继承的,尽管 SQL-99 最终版中忽略了它,但 SQL 标准的未来版本可能会引入。这里基于 SQL-99 标准的草案来讨论问题。

【例 9.52】 假定类型系统支持多重继承,那么可以为助教定义一个类型如下:

```
create type TeachingAssistant
under Student, Teacher
```

TeacherAssistant 将继承 Student 和 Teacher 的所有属性。由于 Name 和 address 属性实际上是从同一个来源即 Person 继承来的,因此同时从 Student 和 Teacher 中继承这两个属性不会引起冲突。但是,一个助教既可能是某个系的学生同时又是另一个系的教师,所以 department 属性在 Student 和 Teacher 中都分别有定义。为了避免两次出现的 department 之间的冲突,我们可以使用 as 子句将它们重新命名,如下面的 TeachingAssistant 类型定义所示:

```
Create type TeachingAssistant
    Under Student with(department as student-dept),
    Teacher with(department as teacher-dept)
```

注意,SQL-99 只支持单继承,即一个类型只能继承一种类型,使用的语法如例 9.50。TeachingAssistant 例子中的多重继承在 SQL-99 中是不支持的。SQL-99 标准在类型定义的尾部还有一个特别的字段,取值为 final 或 not final。其中,关键字 final 表示不能从给定类型创建子类型,not final 表示可以创建子类型。

在 SQL 中,一个结构类型的值必须恰好只有一个“最明确类型(most-specific type)”,即每一个值被创建时必须关联到一个确定的类型,成为它的最明确类型。通过继承,每个值也与它的最明确类型的每个超类型相关联。举例来说,假定一个实体具有类型 Person,同时又具有类型 Student,那么这个实体的最明确类型为 Student,因为 Student 是 Person 的子类型。然而一个实体不能同时既具有类型 Student 又具有类型 Teacher,除非这个实体具有一个如 TeacherAssistant 那样既是 Student 子类型又是 Teacher 子类型的类型。

2. 表继承

SQL-99 中的子表(subtable)对应的是 E-R 概念中的特殊化/一般化。子表的类型必须是父表类型的子类型,因此,父表中的每一个属性均出现在子表中。

【例 9.53】 假设 People 表的定义如下,定义 people 的两个子表 students 和 teachers。

```
create table people of Person
```

定义子表 students 和 teachers:

```
create table students of Student
    under people
create table teachers of Teacher
    under people
```

当我们声明 students 和 teachers 作为 people 的子表时,每一个 students 或 teachers

中出现的元组也隐式地存在于 people 中。如果一个查询用到 people 表,它将查找的不仅是直接插入这个表中的元组,而且还包含插入它的子表(也就是 students 和 teachers)中的元组。但是,只有出现在 people 中的属性才可以被访问。

多重继承也可在表中实现。例如,创建一个类型为 TeachingAssistant 的表:

```
create table teaching-assistants
of TeachingAssistant
under students, teachers
```

作为声明的结果,每一个在 teaching-assistants 中出现的元组也隐式地在表 teachers 和 students 中出现,从而也出现在 people 表中。SQL-99 允许在查询中使用“only people”代替 people 以查询只在 people 中而不在它的子表中的元组。

对于表的一致性要求。如果一个子表和一个父表中的元组的所有继承属性具有同样的值,则称子表中的元组符合(correspond to)父表中的元组。因此,符合的元组表示同一个实体。子表的一致性需求为:

(1) 父表的每个元组至多可以与它的每个直接子表的一个元组符合。

(2) SQL-99 有一个附加的约束,所有符合的元组必须由一个元组派生(插入到一个表中)。

例如,若没有第一个条件,在 students(或 teachers)中就可能有两个元组与同一个人符合。第二个条件排除了 people 中的一个元组分别符合 students 和 teachers 中的一个元组的情况,除非所有这些元组都隐式出现。这是由于一个元组会被插入一个既是 teacher 的子表又是 students 的子表的 teaching-assistants 表中。

由于 SQL-99 不支持多重继承,所以第二个条件实际上防止了一个人既是老师又是学生。即使支持多重继承,在没有子表 teaching-assistants 时这个问题也会出现。显然,建立一个即使没有 teaching-assistants 子表也可以让一个人既是老师又是学生的环境是很有用的。因此,去掉第二个一致性约束是有用的。

子表可以采用无须复制所有继承字段的有效方式进行存储,通常有如下两种方式:

- 每一个表只存储主码(可能是从父表中继承来的)和局部定义的属性。继承属性(主码之外的)不需要存储,因为它可以利用主码与父表的连接得到。
- 每一个表都存储所有继承的和局部定义的属性。插入一个元组时,它仅仅存储在插入的那个表中,可以在其父表中推断它的出现。因为不需要连接,所以访问元组的所有属性会很快。不过,一旦没有第二个一致性约束(即一个实体可能出现在两个子表中而不在它们的公共子表中出现),这种表达将导致信息重复的问题。

9.8.4 引用类型

面向对象的程序设计语言提供了引用对象的能力,一个类型的属性可以是对一个指定类型的对象的引用。

【例 9.54】 定义一个包括 name 字段和 head 字段的 Department 类型,一个 Department 类型的表 departments。其中,head 字段引用到 Person 类型。方法如下所示:

```
create type Department(  
    name varchar(20),  
    head ref(Person) scope people)  
create table departments of Department
```

这里,引用限制在 people 表中的元组。在 SQL-99 中,对指向表的元组的引用范围(scope)的限制是强制的,它使引用的行为与外码类似。

9.8.5 与复杂类型有关的查询

可以从一个简单的例子来看复杂类型的 SQL 查询语言扩展。假设要找出每本书的标题和出版商,下面的查询实现了这项任务:

```
select title, publisher.name  
from books
```

请注意,这里使用点号引用复合属性 publisher 的 name 字段。

1. 路径表达式

在 SQL-99 中,对引用取内容可使用→符号。

【例 9.55】 找出各个部门负责人的名字和地址,查询语句如下:

```
select head→name, head→address  
from departments
```

在上面的查询中,带有→符号的表达式被称为路径表达式(path expression)。由于 head 是一个对 people 表中元组的引用,上述查询中的 name 属性就是 people 表中元组的 name 属性。引用可以用来隐藏连接操作。在上面的例子中,如果没有使用引用,则 departments 的 head 字段就会被声明为 people 表的一个外码。要找出一个部门负责人的姓名和地址,就需要将 departments 与 people 关系显式地做一个连接。因此,使用引用显著地简化了查询。

2. 以集合体为值的属性

怎样处理以集合为值的属性？一个计算集合体值的表达式可以出现在关系名出现的任何地方。

【例 9.56】 根据前面定义的 books 表,查询所有码中包含“database”字样的书,查询语句如下:

```
select title
  from books
 where 'database' in (unnest(keyword-set))
```

在无嵌套关系的 SQL 中,unnest(keyword-set)相当于一个 select-from-where 的子表达式。如果知道一本特定的书具有 3 个作者,我们会这样写:

```
select author-array[1], author-array[2], author-array[3]
  from books
 where title='Database System Concepts'
```

现在,假定想得到一个关系,它包含形式为“书名,作者名”的属性,分别对应每本书和书的每个作者。我们可以使用下面的查询语句:

```
select B.title, A.name
  from books as B, unnest(B.author-array) as A
```

由于 books 的 author-array 属性是一个以集合体为值的字段,因此可以用在需要有一个关系存在的 from 子句中。

3. 嵌套与解除嵌套

将一个嵌套关系转换成具有更少(或没有)关系的过程被称为解除嵌套(unnesting)。books 关系有 author-array 和 keyword-set 两个集合体属性,另外还有 title 和 publisher 两个不是集合体的属性。

【例 9.57】 将 books 关系转化为一个单一的平面关系,使其不包含嵌套关系或者结构类型的属性。我们可以使用以下查询来完成这个任务:

```
select title, A as author, publisher.name as pub-name, publisher.branch as pub-branch,
       K as keyword
  from books as B, unnest(B.author-array) as A, unnest(B.keyword-set) as K
```

from 子句中的变量 B 被声明为以 books 为取值范围,变量 A 被声明为以书 B 的 author-array 中的作者为取值范围,同时 K 被声明为以书 B 的 keyword-set 中的关键字为取值范围。图 9-3 给出了 books 关系的一个实例,图 9-4 显示了上述查询结果形成的 1NF 关系。

将一个 1NF 关系转化为嵌套关系的反向过程称为嵌套(nesting)。嵌套可以作为 SQL 分组操作的一个扩展来完成。在 SQL 分组的常规使用中,要对每个组(逻辑上)创建一个临时的多重集合关系,然后在这个临时关系上应用一个聚集函数。可以通过返回这个多重集合而不应采用聚集函数的方式创建一个嵌套关系。

假定有一个如图 9-3 所示的 1NF 关系 flat-books,下面的查询在属性 keyword 上对关系进行了嵌套:

```
select title, author, Publisher(pub-name, pub-branch) as publisher,
       set(keyword) as keyword-set
from flat-books
groupby title, author, publisher
```

在 flat-books 关系上执行上述查询的结果如图 9-5 所示。如果要同时对作者属性进行嵌套,将图 9-3 中的 1NF 表 flat-books 转化为图 9-2 所示的嵌套表 books,我们可以使用如下查询:

```
select title, set(author) as author-set, Publisher(pub-name, pub-branch) as publisher,
       set(keyword) as keyword-set
from flat-books
groupby title, publisher
```

title	author	publisher	keyword-set
		(pub-name, pub-branch)	
Compilers	Smith	(McGraw-Hill, New York)	{ parsing, analysis }
Compilers	Jones	(McGraw-Hill, New York)	{ parsing, analysis }
Networks	Jones	(Oxford, London)	{ Internet, Web }
Networks	Frick	(Oxford, London)	{ Internet, Web }

图 9-5 flat-books 关系的一个部分嵌套版本

另一种创建嵌套关系的方法是在 select 语句中使用子查询。下面的查询阐明了该方法,它与上面的查询执行了同样的任务。

```
Select title,
       (select author
        from flat-books as M
        where M. title=O. title) as author-set,
       Publisher(pub-name, pub-branch) as publisher,
       (select keyword
        from flat-books as N
```

```

        where N. title=O. title) as keyword-set,
    from flat-books as O

```

系统对 select 子句外部查询中的 from 和 where 子句生成的每一个元组执行嵌套子查询。外部查询中的 O. title 属性被用到嵌套查询中,以确保对每一本书的题目产生正确的作者和关键字集合。这种方法的优点在于可以在嵌套查询上使用 orderby 子句生成一个有序的结果。数组或列表可以从嵌套查询的结果中构造出来。如果没有排序,数组和列表将不能惟一确定。

9.8.6 函数和过程

SQL-99 允许定义函数、过程和方法。定义可以通过 SQL-99 有关过程的组件,也可以通过外部程序设计语言实现,例如 Java、C 或 C++。我们首先看 SQL-99 中的定义,然后了解如何使用外部语言中的定义。有些数据库系统支持它们自己的过程语言,例如 Oracle 中的 PL/SQL 和 Microsoft SQL Server 中的 TransactSQL。它们类似于 SQL-99 的有关过程部分,但在语法和语义上有所区别。详细信息可参见各自的系统手册。

1. SQL 函数和过程

假定我们想要这样一个函数:给定一个书名,返回其作者数,使用 4NF 模式。我们可以用下面的方式定义这个函数:

```

create function author-count(title varchar(20))
returns integer
begin
declare a-count integer;
    select count(author) into a-count
    from authors
    where authors. title=title
return a-count;
end

```

这个过程可以用在一个返回具有多个作者的所有书名的查询:

```

select title
from books4
where author-count(title)>1

```

对于特定的数据类型(比如图像和几何对象)来说函数特别有用。例如,用在地图数据库中的一个多边形数据类型可能有一个用于判断两个多边形是否重叠的函数,一个图像数据类型可能有一个用于比较两幅图的相似性的函数。函数可以用外部语言(比如 C)

来编写。一些数据库系统也支持返回关系(即元组的多重集合)的函数,尽管 SQL-99 并不支持这些函数。

前面介绍的方法可以看作与结构类型相关联的函数。它们的第一个参数 `self` 是隐含的,设置为调用这个方法的结构类型的值。因此,方法主体可以通过 `self.a` 来引用这个值的属性 `a`。这些属性也可以被该方法更新。

SQL-99 也支持过程。`author-count` 函数也可以写成一个过程:

```
Create procedure author-count-proc(in title varchar(20), out a-count integer)
begin
    select count(author) into a-count
    from authors
    where authors.title=title
end
```

可以在一个 SQL 过程中或者嵌入式 SQL 中使用 `call` 语句调用过程:

```
declare a-count integer;
call author-count-proc('Database Systems Concepts', a-count);
```

SQL-99 允许多个过程同名,只要同名过程的参数个数不同即可。名称和参数个数用于标识一个过程。SQL-99 也允许多个函数同名,只要这些同名函数的参数个数不同,或者,即使参数个数相同,但至少有一个参数的类型不同即可。

2. 外部语言程序

SQL-99 允许使用程序设计语言(如 C 或 C++)定义函数。这种方式定义的函数比 SQL 中定义的函数效率更高,无法在 SQL 中执行的计算可以利用这样的函数实现。

【例 9.58】 在一个元组的数据上执行复杂数学计算。外部过程和函数可以这样指定:

```
create procedure author-count-proc(in title varchar(20), out count integer)
language C
external name '/usr/avi/bin/author-count-proc'
create function author-count (title varchar(20))
returns integer
language C
external name '/usr/avi/bin/author-count'
```

外部语言过程需要处理空值和异常。因此它们必须具有几个额外的参数:一个指明失败/成功状态的 `sqlstate` 值,一个存储函数返回值的参数,一些指明每个参数/函数结果的值是否为空的指示器变量。如果在声明语句的上方额外添加一行 `parameter style`

general,指定外部过程/函数只使用说明的变量并且不处理空值和异常。

3. 过程的构造

SQL-99 支持多种过程的构造,这使之具有与通用程序设计语言相当的几乎所有的功能。SQL-99 标准中处理这种构造的部分称为持久存储模块 (persistent stroage module, PSM)。

1) while 语句和 repeat 语句

一个复合语句有 begin ... end 的形式,在 begin 和 end 之间会包含多条 SQL 语句。SQL-99 支持 while 语句和 repeat 语句,语法如下:

```
declare n integer default 0;
while n<10 do
    set n=n+1;
end while
repeat
    set n=n-1;
until n=0
end repeat
```

这段代码没什么用,只是说明 while 和 repeat 循环的语法。

2) for 语句

for 循环允许对查询的所有结果迭代:

```
declare n integer default 0;
for r as
    select balance from account
        where branch-name='Perryridge'
do
    set n=n+r.balance
end for
```

程序在 for 循环开始执行的时候隐式地打开一个游标,每次用它获得一个行的值,并存入 for 循环变量(如上面例子中的 r)中。可以赋予游标一个名称,在关键字 as 后插入 cn cursor for 即可完成,其中的 cn 就是游标的名称。游标名可用于对该游标指向的元组进行更新/删除操作。语句 leave 可用来退出循环,而 iterate 表示跳过剩余语句从循环的开始进入下一个元组。

3) if-then-else 语句与 case 语句

条件语句包括 if-then-else 语句,其语法如下:

```

if r. balance < 1000
    then, set l = l + r. balance
elseif r. balance < 5000
    then set m = m + r. balance
else set h = h + r. balance
end if

```

这段代码假定 l, m 和 n 是整型变量, r 是行变量。如果我们用这段 if-then-else 代码替换前面的 for 循环中的“set $n = n + r. balance$ ”行, 则循环将按低、中和高余额分别计算这 3 类账户余额的总数。

SQL-99 的 case 语句类似于 C/C++ 语言中的 case 语句。

SQL-99 还包括发信号通知异常条件(exception condition), 以及声明句柄(handler)来处理异常的概念。代码如下:

```

declare out-of-stock condition
declare exit handler for out-of-stock
begin
...
end

```

begin 和 end 之间的语句可以通过执行 signal out-of-stock 来引发一个异常。这个句柄说明, 如果条件发生, 将会采取动作以终止 begin end 中的语句。替换的动作将是 continue, 它继续从引发异常的语句的下一条语句开始执行。除了明确定义的条件, 还有一些预定义的条件, 比如 sqlexception、sqlwarning 和 not found。

图 9-6 是有关 SQL-99 过程构造的实例。过程 findEmpl 计算给定经理(由参数 mgr 指定)的所有直接和非直接管理的员工的集合, 并且将结果集中员工的名字存储在一个称为 empl 的关系中(假定这个关系已经存在)。关系 manager(empname, mgrname)(假定可用)指明了哪个员工直接为哪个经理工作。所有的直接/非直接管理的员工的集合构成了 manager 关系的传递闭包的主体。

该过程使用了两个临时表: newemp 和 temp。该过程在 repeat 循环之前把所有直接为 mgr 工作的员工插入到 newemp 中。repeat 循环首先把 newemp 中的所有员工添加到 empl 中。然后, 它计算为 newemp 中的人服务的员工(除了那些已经找到的 mgr 的员工), 并把它们存储到临时表 temp 中。最后用 temp 中的内容替换 newemp 中的内容。repeat 循环在找不到新员工(非直接的)时终止。请注意 except 子句的用处, 它确保即使在有管理关系循环(反常)的情况下, 该过程仍可工作。例如, 如果 a 为 b 工作, b 为 c 工作, 而 c 又为 a 工作, 就出现了循环。

虽然在管理控制中循环不可能出现,但是其他应用中可能会出现循环。例如,假设有关系 flights(to, from),表示从一个城市直飞可达的其他城市。可以修改 findEmple 过程来查找从给定城市利用一次或多次航班的一个序列能到达的所有城市。所要做的就是用 flight 替换 manager,并且替换相应的属性名。这种情况下有可能出现一个可到达性的循环,但由于该过程排除了已经找到的城市,所以它能正确执行。

```
create procedure findEmple(in mgr char(10))
--找出所有直接或间接为 mgr 工作的员工,并将他们加到关系 empl(name)中。
--关系 manager(empname, mgrname)指出谁直接为谁工作。
begin
    create temporary table newemp (name char(10));
    create temporary table temp (name char(10));
    insert into newemp
        select empname
        from manager
        where mgrname=mgr
    repeat
        insert into empl
            select name
            from newemp;

        insert into temp
            (select manager.empname
             from newemp, manager
             where newemp.empname=manager.mgrname;
            )
        except (
            select empname
            from empl
            );
        delete from newemp;
        insert into newemp
            select *
            from temp;
        delete from temp;

    until not exists (select * from newemp)
    end repeat;
End
```

图 9-6 找出一个经理管理的所有员工

第 10 章 系统开发与运行

10.1 软件工程和软件开发项目管理知识

自从 1968 年首次提出“软件工程”这个概念以来,软件工程已经成为计算机软件的一个重要分支和研究方向。软件工程是指应用计算机科学、数学及管理科学等原理,以工程化的原则和方法来解决软件问题的工程。其目的是提高软件生产率,提高软件质量,降低软件成本。

软件工程涉及到软件开发、维护和管理等多方面的原理、方法、工具与环境,限于篇幅本章不能对软件工程做全面的介绍。根据数据库系统工程师级考试大纲的要求,本章着重介绍软件开发过程中的原理,其他内容只做简单的介绍。

10.1.1 软件工程概述与软件生存周期

1. 软件工程概述

早期的软件主要指程序,程序的开发采用个体工作方式,开发工作主要依赖于开发人员的个人技能和程序设计技巧。当时的软件通常缺少与程序有关的文档,软件开发的实际成本和进度往往与预计的相差甚远,软件的质量得不到保证,开发出来的软件常常不能使用户满意。随着计算机应用的需求不断增长,软件的规模也越来越大,然而软件开发的生生产率远远跟不上计算机应用的迅速增长。此外,由于软件开发时缺少好的方法指导和工具辅助,同时又缺少相关文档,使得大量已有的软件难以维护。上述这些问题严重地阻碍了软件的发展。20 世纪 60 年代中期,人们把上述的软件开发和维护过程中所遇到的各种问题称为“软件危机”。

1968 年在德国召开的 NATO(North Atlantic Treaty Organization,北大西洋公约组织)会议上,首次提出了“软件工程”这个名词,希望用工程化的原则和方法来克服软件危机。在此以后,人们开展了软件开发模型、开发方法、工具与环境的研究,提出了瀑布模型、演化模型、螺旋模型和喷泉模型等开发模型,出现了面向数据流方法、面向数据结构的方法和面向对象方法等开发方法,以及一批计算机辅助的软件工程(computer aided software engineering, CASE)工具和环境。

2. 软件生存周期(life cycle)

同任何事物一样,一个软件产品或软件系统也要经历孕育、诞生、成长、成熟以及衰亡

等多个阶段,一般称为软件生存周期。根据这一思想,把上述的基本过程进一步展开,可以得到软件生存周期的6个工作阶段,即计划制定、需求分析、设计、程序编制、测试以及运行维护。以下对这6个阶段的任务做一概括性的描述。

(1) 软件项目计划(planning)。确定待开发软件系统的总目标,对其进行可行性分析,并对资源进行分配。对进度安排等做出合理的计划。软件项目计划阶段的参加人员有用户、项目负责人和系统分析员。该阶段产生的文档有可行性分析报告和项目计划书。

(2) 需求分析和定义(requirement analysis and definition)。确定待开发软件系统的功能、性能、数据以及界面等要求,从而确定系统的逻辑模型。该阶段的参加人员有用户、项目负责人和系统分析员。产生的文档有需求规格说明书(requirements specification)。

(3) 软件设计(software design)。软件设计是软件工程的技术核心。软件设计可以分为概要设计和详细设计。概要设计的任务是模块分解,确定软件的结构、模块的功能和模块间的接口,以及设计全局数据结构。详细设计的任务是设计每个模块的实现细节和局部数据结构。概要设计阶段参加的人员有系统分析员和高级程序员,详细设计阶段参加的人员有高级程序员和程序员。设计阶段产生的文档有设计规格说明书(design specification),它也可以分为概要设计说明书和详细设计说明书。根据需要还可产生数据说明书和模块开发卷宗。

(4) 编码(coding)。编码的任务是用某种程序语言为每个模块编写程序。编码阶段参加的人员有高级程序员和程序员,产生的文档是源程序清单。

(5) 测试(testing)。测试是保证软件质量的重要手段,其主要方式是在设计测试用例的基础上检验软件的各个组成部分。测试阶段的参加人员通常由另一部门(或单位)的高级程序员或系统分析员承担,该阶段产生的文档有软件测试计划和软件测试报告。

(6) 运行/维护(running/maintenance)。已交付的软件正式投入使用,便进入运行阶段。这一阶段可能持续若干年甚至几十年。在运行过程中,由于多方面的原因,可能需要对软件进行修改。例如:运行中发现了软件中的错误需要修正;为了适应变化了的软件工作环境,须做适当变更;为了增强软件的功能需做调整。

10.1.2 软件开发项目管理基础知识

为了使软件开发项目获得成功,必须对软件开发项目的工作范围、可能遇到的风险、需要的资源(人、软硬件)、要实现的任务、经历的里程碑、花费的工作量(成本),以及进度的安排等做到心中有数。而软件项目管理可以提供这些信息。这种管理始于技术工作开始之前,在软件从概念到实现的过程中持续进行,最后终止于软件工程过程结束。

1. 成本估算

由于软件具有可见性差、定量化难等特殊性质,因此很难在项目完成前准确地估算出开

发软件所需的工作量和费用。通常可以根据以往开发类似软件的经验(也可以是别人的经验)进行成本估算。也可以将软件项目划分成若干个子系统或按照软件生存周期的各个阶段分别估算其成本,然后汇总出整个软件的成本。此外还可以使用经验公式和成本估算模型进行估算。

一种常用的成本估算是先估计完成软件项目所需的工作量(人月数),然后根据每个人月的代价(金额)计算软件的开发费用:

$$\text{开发费用} = \text{人月数} \times \text{每个人月的代价}$$

另一种方法是估计软件的规模(通常指源代码行数),然后根据每行源代码的平均开发费用(包括分析、设计、编码和测试所花的费用),计算软件的开发费用:

$$\text{开发费用} = \text{源代码行数} \times \text{每行平均费用}$$

估算源代码行数时,可以请 n 若干有经验的专家,每位专家对软件给出三个估计值:

- a_i ——最少源代码行数(该软件可能的最小规模);
- b_i ——最大源代码行数(该软件可能的最大规模);
- m_i ——最可能的源代码行数(该软件最可能的规模)。

然后计算出每位专家的估算期望值 $E_i = \frac{a_i + 4m_i + b_i}{6}$, n 位专家的估算期望值的平

均值 $\frac{1}{n} \sum_{i=1}^n E_i$ 就是代码行数的估算值。

典型的成本估算模型有普特南(Putnam)模型和 CoCoMo(Constructive Cost Model)模型等。限于篇幅,本节不做详细介绍。

这些估算模型已经用软件实现,称为自动估算工具。这种自动估算工具使得管理或计划人员能够估算待开发软件项目的成本和工作量,还可以对人员配置和交付日期等进行估计。Gordon 集团的 BYL(Before You Leap)、Wang 研究所的 WICOMO(Wang Institute Cost Model)及 DEC 公司的 DECPlan 都是基于 CoCoMo 模型的自动估算工具。SLIM 是基于软件生存期中的 Rayleigh-Norden 曲线和 Putnam 估算模型的一种自动成本估算工具。限于篇幅,本节不做详细介绍。

2. 风险分析

罗伯特·查瑞特(Robert Charette)在他关于风险分析和驾驭的书中对风险的概念给出了如下的定义:“首先,风险关系到未来发生的事情。……问题是,能否通过改变今天的活动为我们的明天创造一个完全不同的充满希望的美好前景。其次,风险会发生变化,就像爱好、意见、动作或地点会变化一样……第三,风险导致选择,而选择本身将带来不确定事件。因此风险就像死亡那样,是一个其生命很少确定性的东西。”在软件工程的环境中考虑风险时,应主要基于查瑞特提出的这 3 个概念。一是关心未来,风险是否会导致软

件项目失败？二是关系变化，在用户需求、开发技术、目标机器以及其他所有与项目有关的实体中会发生什么变化？三是必须解决选择问题：应当采用什么方法和工具，应当配备多少人力，在质量上强调到什么程度才能满足要求？

风险分析实际上包括 4 个不同的活动：风险识别、风险预测、风险评估和风险控制。

3. 进度管理

进度的合理安排是如期完成软件项目的重要保证，也是合理分配资源的重要依据。因此，进度安排是管理工作的一个重要组成部分。软件开发项目的进度安排有两种方式：

- 系统最终交付日期已经确定，软件开发部门必须在规定期限内完成；
- 系统最终交付日期只确定了大致的年限，最后交付日期由软件开发部门确定。

进度安排的常用图形描述方法有 Gantt 图(甘特图)和 PERT(program evaluation & review technique, 计划评审技术)图。

1) Gantt 图

Gantt 图中横坐标表示时间(如时、天、周、月和年等)，纵坐标表示任务。图中的水平线段表示任务的进度安排，线段的起点和终点对应于横坐标上的时间分别表示该任务的开始时间和结束时间，线段的长度表示完成该任务所需的时间。图 10-1 所示的 Gantt 图描述了 3 个任务的进度安排。该图表示：任务 1 首先开始，完成它需要 12 周时间；任务 2 在两周后开始，完成它需要 18 周；任务 3 在 12 周后开始，需要 10 周。

Gantt 图能清晰地描述每个任务从何时开始、到何时结束以及各个任务之间的并行性。但是它不能清晰地反映出各任务之间的依赖关系，难以确定整个项目的关键所在，也不能反映出计划中有潜力的部分。

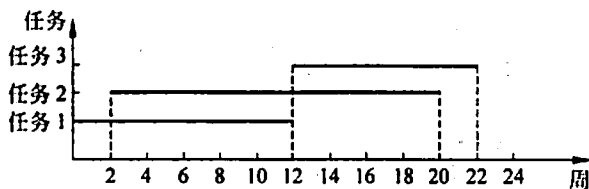


图 10-1 Gantt 图实例

2) PERT 图

PERT 图是一个有向图，图中的箭头表示任务，可以标上完成该任务所需的时间。图中的结点表示流入结点的任务的结束，流出结点的任务的开始，我们把结点称为事件。只有当流入该结点的所有任务都结束时，结点所表示的事件才出现，流出结点的任务才可以开始。事件本身不消耗时间和资源，它仅表示某个时间点。一个事件有一个事件号和出现该事件的最早时刻和最迟时刻。最早时刻表示在此时刻之前从该事件出发的任务不可

能开始。最迟时刻表示从该事件出发的任务必须在此时刻之前开始,否则整个工程就不能如期完成。每个任务还可以有一个松弛时间(slack time),表示在不影响整个工期的前提下,完成该任务有多少机动余地。为了表示任务间的关系,图中还可以加入一些空任务(用虚线箭头表示),完成空任务的时间为0。PERT图的基本符号如图10-2所示。

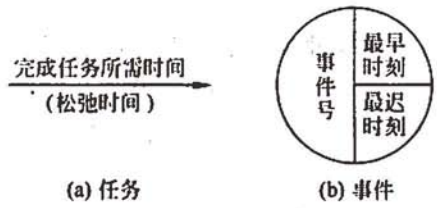


图 10-2 PERT图的基本符号

图10-3是PERT图的一个实例。不难看出,图10-3中松弛事件为0的这些任务是完成整个工程的关键路径,其事件流为:1—>2—>3—>4—>6—>8—>10—>11。

PERT图不仅给出了每个任务的开始时间、结束时间和完成该任务所需的时间,还给出了任务之间的关系,即哪些任务完成后才能开始另执行一些任务,以及如期完成整个工程的关键路径。图中的松弛时间则反映了完成某些任务时可以推迟其开始时间或延长其所需完成的时间。但PERT图不能反映任务之间的并行关系。

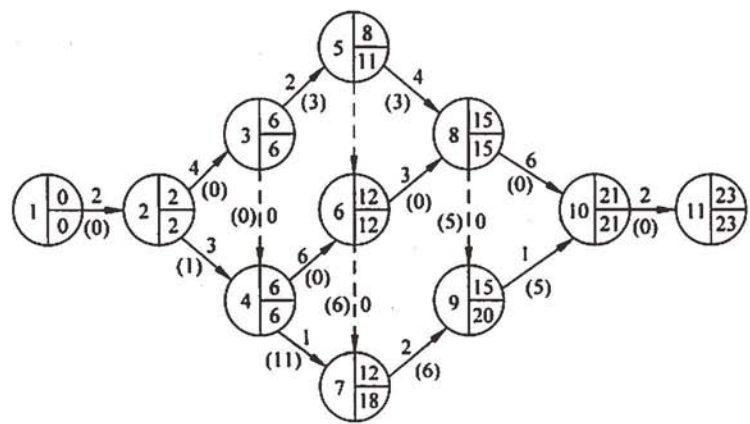


图 10-3 PERT图实例

4. 人员管理

合理地组织好参加软件项目的人员,充分发挥每个人的作用,有利于软件项目的成功开发。在组织人员时,应考虑软件项目的特点和软件人员的素质等多方面的因素。

可以按软件项目对软件人员分组,如需求分析组、设计组、编码组、测试组和维护组

等。为了控制软件的质量,还可以有质量保证组。

程序设计小组的组织形式也可以有多种,如主程序员组、无主程序员组以及层次式程序员组等。

(1) 主程序员组:主程序员组由一名主程序员、一名后备程序员(back up programmer)、一名资料员和若干名程序员组成。主程序员由经验丰富、能力强的高级程序员担任,他是该组织的技术领导和项目负责人,全面负责软件项目的开发。后备程序员是主程序员的助手,协助主程序员工作,必要时能代替主程序员工作。资料员负责保存和管理所有的软件配置元素,如文档资料、程序清单和存储介质等。程序员则负责完成主程序员分配给他的任务。这种组织形式便于集中领导,统一步调,容易按规范办事,但不利于发挥个人的积极性。

(2) 无主程序员组:无主程序员组中的成员之间相互平等,工作目标和决策都由全体成员民主讨论,根据需要也可以轮流坐庄。这种组织形式民主气氛比较浓,依赖个人的成分少,有利于发挥每个人的积极性,但缺点是往往职责不明确,出了问题谁也不负责,不利于与外界的联系。

(3) 层次式程序员组:组中有一位组长,组长负责全面的工作,他领导若干名高级程序员,每个高级程序员又领导若干名程序员。这种组织形式适合于具有层次结构特点的软件项目,该项目可分成若干个子项目,每个高级程序员负责一个子项目,然后再对子项目分解,并分配给程序员。

10.1.3 软件开发方法

软件开发可以使用许多种不同的方法,本节简要介绍 3 种软件开发方法:结构化方法、面向对象方法和原型法。

1. 结构化方法

结构化方法是结构化分析(structured analysis, SA)和结构化设计(structured design, SD)的总称。结构化方法是目前最成熟、应用最广泛的信息系统开发方法之一,它的优点是有一套严格的开发程序,各开发阶段都要求有完整的文档记录,国内外已有许多成功开发的例子。

2. 面向对象方法

面向对象方法起源于 20 世纪 60 年代由挪威计算中心开发的 Simula 67 语言,它首先引入了类的概念和继承机制。20 世纪 80 年代美国加州 Xerox 研究中心推出的 Smalltalk 语言和环境,使面向对象方法得到比较完善的实现,掀起了面向对象方法研究的高潮。20 世纪 80 年代至 20 世纪 90 年代,涌现出大批实用的面向对象语言,如 C++、Object Pascal 和 Eiffel 等,大大提高了软件开发的效率以及软件的可复用性和可维护性。

20 世纪 90 年代面向对象方法不再局限于编程阶段,而是朝着软件生存周期的前期阶段发展,形成了面向对象的分析和设计,发展成为一整套软件方法学,成为计算机领域的主流技术之一。

面向对象方法的基本思想是从现实世界中客观存在的事物出发来构造软件系统的。软件系统适用的业务范围称作软件的问题领域,把问题领域中事物的特征抽象地描述成类,由类建立的对象作为系统的基本构成单位,它们的内部属性与服务描述了客观存在的事物的静态特征和动态特征。对象类之间的继承关系、聚集关系、消息和关联反映了问题领域中事物之间实际存在的各种关系。

3. 原型法

软件系统的开发通常采用结构化方法。结构化方法要求严格定义或预先明确说明用户需求。但系统开发的实践表明,需求会经常发生变化。

原型化方法的思想是,在获得一组基本的需求后,快速地加以“实现”。随着用户或开发人员对系统理解的加深而不断地对这些需求进行补充和细化。系统的定义是在逐步发展的过程中进行的,而不是一开始就预见一切。原型化方法是一种确定需求的策略,对用户的需求进行抽取、描述和求精。它快速地迭代并建立最终系统的工作模型,对问题的定义采用启发式的方式,并由用户做出响应,是一种动态定义技术。

10.1.4 软件工具与软件开发环境

用来辅助软件开发、运行、维护、管理和支持等过程中的软件称为软件工具。早期的软件工具主要用来辅助程序员编程,如编辑程序、编译程序及排错程序等。在提出了软件工程的概念以后,又出现了软件生存周期的概念,出现了许多开发模型和开发方法,软件管理也开始受到人们的重视。与此同时,出现了一批软件工具来辅助软件工程的实施,这些软件工具涉及到软件开发、维护和管理过程中的各项活动,并辅助这些活动高效、高质量地进行。因此,软件工具通常也称为 CASE(计算机辅助软件工程,Computer Aided Software Engineering)工具。

软件开发环境是指支持软件产品开发的软件系统,它由软件工具集和环境集成机制构成。工具集应包括支持软件开发过程、活动和任务的软件工具,以对软件开发提供全面的支持。环境集成机制为工具集成和软件开发、维护及管理提供统一的支持,它通常包括数据集成、控制集成和界面集成。

1. 软件工具

软件工具的种类繁多,按照软件过程的活动可以划分为支持软件开发过程的工具,支持软件维护过程的工具,以及支持软件管理过程的工具等。

1) 软件开发工具

对应于软件开发过程的各种活动,软件开发工具通常有需求分析工具、设计工具、编码与排错工具,以及测试工具等。

2) 软件维护工具

辅助软件维护过程的软件称为软件维护工具,它辅助维护人员对软件代码及其文档进行各种维护活动。软件维护工具主要有版本控制工具、文档分析工具、开发信息库工具、逆向工程工具和再工程工具。

3) 软件管理和软件支持工具

软件管理和软件支持工具用来辅助管理人员和软件支持人员的管理活动和支持活动,以确保高质量地完成软件开发。辅助软件管理和软件支持的工具很多,其中常用的工具有项目管理工具、配置管理工具和软件评价工具。

4) 软件开发工具的评价和选择

现在,市场上的各类软件开发工具十分丰富,有免费的,有价格便宜的,也有昂贵的。如何评价和选择适合本人、本单位和本项目的软件开发工具呢?可以根据以下标准来衡量软件开发工具的优劣:

- 功能;
- 易用性;
- 稳健性;
- 硬件要求和性能;
- 服务和支持。

2. 软件开发环境

软件开发环境(software development environment)是支持软件产品开发的软件系统。它由软件工具集和环境集成机制构成,前者用来支持软件开发的相关过程、活动和任务,后者为工具集成和软件开发、维护和管理提供统一的支持,它通常包括数据集成、控制集成和界面集成。通过环境集成机制,各工具用统一的数据接口规范存储或访问环境信息库。各工具采用统一的界面形式,保证各工具界面的一致性,同时为各工具或开发活动之间的通信、切换、调度和协同工作提供支持。在软件开发环境中进行软件开发,可以使用环境中提供的各种工具。同时在环境信息库的支持下,一个工具所产生的结果信息可以为其他工具利用,使得软件开发的各项活动得到连续的支持,从而大大提高了软件的开发效率,提高了软件的质量。

软件开发环境的特征是:

- 环境的服务是集成的。软件开发环境应支持多种集成机制,如平台集成、数据集成、界面集成、控制集成和过程集成等;
- 环境应支持小组工作方式,并为其提供配置管理;

- 环境的服务可用于支持各种软件开发活动,包括分析、设计、编程、测试、调试和文档等。

集成型开发环境是一种把支持多种软件开发方法和开发模型的软件工具集成在一起的软件开发环境。这种环境应该具有开放性和可剪裁性。开放性为环境外的工具集成到环境中来提供了方便,可剪裁性能够根据不同的应用和不同的用户需求进行剪裁,以形成特定的开发环境。

10.1.5 软件质量管理与质量保证

软件质量是指反映软件系统或软件产品满足规定或隐含需求的能力的特征和特性全体。软件质量保证是指为保证软件系统或软件产品充分满足用户要求的质量而进行的有计划、有组织的活动,其目的是生产高质量的软件。

1. 软件质量特性

讨论软件质量首先要了解软件的质量特性。目前已经有多种软件质量模型来描述软件质量特性。

1) ISO/IEC 9126 软件质量模型

ISO/IEC 9126 软件质量模型由 3 个层次组成:第一层是质量特性;第二层是质量子特性;第三层是量度指标。该模型的质量特性和质量子特性如图 10-4 所示。



图 10-4 ISO/IEC 软件质量模型

其中各质量特性和质量子特性的含义如下:

(1) 功能性(functionality):与一组功能及其指定的性质的存在有关的一组属性。功能是指满足规定或隐含需求的那些功能。

- 适应性(suitability):与对规定任务能否提供一组功能以及这组功能是否适合有关的软件属性。

- 准确性(accurateness): 与能够得到正确或相符的结果或效果有关的软件属性。
- 互用性(interoperability): 与同其他指定系统进行交互操作的能力相关的软件属性。
- 依从性(compliance): 使软件服从有关的标准、约定、法规及类似规定的软件属性。
- 安全性(security): 与避免对程序及数据的非授权、故意或意外访问的能力有关的软件属性。

(2) 可靠性(reliability): 与在规定的时间内和规定的条件下, 软件维持在其性能水平有关的能力。

- 成熟性(maturity): 与由软件故障引起失效的频度有关的软件属性。
- 容错性(fault tolerance): 与在软件错误或违反指定接口的情况下, 维持指定的性能水平的能力有关的软件属性。
- 易恢复性(recoverability): 与在故障发生后, 重新建立其性能水平并恢复直接受影响数据的能力, 以及为达到此目的所需的时间和努力有关的软件属性。

(3) 易使用性(usability): 与为使用所需的努力和由一组规定或隐含的用户对这样使用所做的个别评价有关的一组属性。

- 易理解性(understandability): 与用户为理解逻辑概念及其应用所付出的劳动有关的软件属性。
- 易学性(learnability): 与用户为学习有关应用(例如操作控制、输入、输出)所付出的努力相关的软件属性。
- 易操作性(operability): 与用户为进行操作和操作控制所付出的努力有关的软件属性。

(4) 效率(efficiency): 在规定条件下, 软件的性能水平与所用资源量之间的关系有关的软件属性。

- 时间特性(time behavior): 与响应和处理时间以及软件执行其功能时的吞吐量有关的软件属性。
- 资源特性(resource behavior): 与软件执行其功能时所使用的资源量以及使用资源的持续时间有关的软件属性。

(5) 可维护性(maintainability): 与进行规定的修改所需要的努力有关的一组属性。

- 易分析性(analyzability): 与为诊断缺陷或失效原因及为判定待修改部分所需努力有关的软件属性。
- 易改变性(changeability): 与进行修改、排错或适应环境变换所需努力有关的软件属性。

- 稳定性(stability): 与修改造成未预料效果的风险有关的软件属性。
- 易测试性(testability): 与确认已修改软件所需努力有关的软件属性。

(6) 可移植性(portability): 与软件可从某一环境转移到另一环境的能力有关的一组属性。

- 适应性(adaptability): 与无须采用有别于为该软件准备的处理或手段就能适应不同的规定环境有关的软件属性。
- 易安装性(installability): 与在指定环境下安装软件所需努力有关的软件属性。
- 一致性(conformance): 使软件服从与可移植性有关的标准或约定的软件属性。
- 易替换性(replaceability): 在该软件环境中用来替代指定的其他软件的可能和努力有关的软件属性。

2) Mc Call 软件质量模型

Mc Call 软件质量模型从软件产品的运行、修正和转移等 3 个方面确定了 11 个质量特性(见图 10-5)。Mc Call 也给出了一个 3 层模型框架,第一层是质量特性,第二层是评价准则,第三层是量度指标。



图 10-5 Mc Call 软件质量模型

2. 软件质量保证概述

软件质量保证是指为保证软件系统或软件产品充分满足用户要求的质量而进行的有计划、有组织的活动,其目的是生产高质量的软件。在软件质量方面需要强调 3 个要点:首先,软件必须满足用户规定的需求,与用户需求不一致的软件,就无质量可言;其次,软件应遵守规定标准所定义的一系列开发准则,不遵循这些准则的软件,其质量难以得到保证;第三,软件还应满足某些隐含的需求,例如有好的可理解性、可维护性等,而这些隐含的需求可能未被明确地写在用户规定的需求中,如果软件只满足它的显式需求而不满足其隐含需求,那么该软件的质量是令人置疑的。

软件质量保证包括与以下 7 个主要活动相关的各种任务。

1) 应用技术方法

应该把软件质量设计到软件产品或软件系统中去,而不是在事后再施加质量保证。由于这个原因,软件质量保证首先从选择一组技术方法和工具开始,这些方法和工具帮助分析人员形成高质量的规格说明和高质量的设计。

2) 进行正式的技术评审

一旦形成了规格说明和设计,就要对它们进行质量评估。完成质量评估的中心活动是正式的技术评审。正式的技术评审是一种由技术人员实施的程式化活动,其惟一的目的是发现质量问题。在多数情况下,评审能像测试一样有效地发现软件中的缺陷。

3) 软件测试

软件测试组合了多种测试策略,这些测试策略带有一系列有助于有效地检测错误的测试用例设计方法。许多软件开发人员把软件测试当作质量保证的“安全网”,也就是说,开发人员以为通过测试能发现很多错误,借此减轻对其他软件质量保证活动的需要。遗憾的是,即使是完成得很好的测试也不会像人们所期望的那样发现所有的错误。

4) 标准的实施

各单位在软件工程过程中采用正式开发标准和过程的程度是不同的。在多数情况下,标准由客户确定,或者根据某些章程确定;某些场合下标准是自己确定的。如果存在正式的标准,软件质量保证活动必须保证遵循这些标准。与标准是否一致的评估可以作为正式技术评审的一部分来进行。

5) 控制变更

软件质量的主要威胁来自“变更(change)”。对软件的每次变更都有可能引入错误或者引起传播错误的副作用。控制变更过程即通过对变更的正式申请过程的控制、评价变更的特性和控制变更的影响来直接影响软件的质量。变更控制应用软件开发期间和较后的软件维护阶段。

6) 量度(metrics)

量度是任何工程科学必备的活动。软件质量保证的重要目标是跟踪软件质量和评价方法学及程序上的变更对软件质量的影响程度。要达到这个目标,必须实施软件量度。软件量度包括技术上的和面向管理的量度。

7) 记录保存和报告

记录保存和报告是为软件质量保证提供、收集和传播软件质量保证信息的活动。评审、检查、变更控制、测试和其他软件质量保证活动的结果必须变成项目历史记录的一部分,并且应该把它传播给需要知道这些结果的开发人员。例如记录过程设计的每次正式技术评审结果,并把记录放置在相应的文件夹中,该文件夹应包含有关模块的所有技术信

息和软件质量保证信息。

10.1.6 软件过程能力评估

软件产品的质量取决于软件开发过程,具有良好软件过程的软件机构能够开发出高质量的软件产品。这一点虽然早已为人们公认,但切实地在软件过程方面开展工作也只是十多年前的事。1987 年在美国国防部支持下,卡内基·梅隆大学率先推出了软件工程评估项目的研究成果——软件过程能力成熟度模型 CMM。很快就引起了软件界的广泛关注,并引发了一系列反响,并在其基础上形成了国际标准(ISO/IEC 15504)。

1. 软件过程评估的意义

软件过程评估是软件改进和软件能力评价的前提。

1) 软件过程改进的需要

(1) 软件过程不断改进是软件工程的基本原理之一。1983 年美国 TRW 公司的 B. W. Boehm 总结了该公司在 12 年内总共花费 15 000 人年,先后开发 5 代指挥控制软件的经验,得出了以下七条原则:

- 按软件生存周期分阶段指定计划并认真实施。
- 逐阶段进行确认。
- 坚持严格的产品控制。
- 使用现代程序设计技术。
- 明确责任。
- 用人少而精。
- 不断改进开发过程。

这就是著名的软件工程七原理。由此可见,不断改进软件开发过程是软件工程的基本原理之一。

(2) 软件过程改进是软件生存周期的基本过程之一。软件工程界始终十分重视软件过程的研究。20 世纪 70 年代中期形成了软件生存周期的概念,1995 年正式发布了一项国际标准,即 ISO/IEC 12207 信息技术——软件生存周期过程,这是软件过程研究的一个重要成果。这项标准科学地定义了软件生存周期的过程,总共 17 个,其中一个就是过程改进。

2) 降低软件风险的需要

(1) 软件采购者的需要:软件产品或软件服务的采购单位通过招标选择承办者时,为了降低风险,需要对备选单位的软件过程能力进行评价,而这种评价的依据是对该单位软件过程的评估结果。

(2) 软件研制者的需要:软件产品研制单位和软件服务单位在根据顾客的需求、进

行投标时,为了降低风险,需要对自己的软件过程能力进行评价,避免承担力所不能及的任务,而这种评价的依据仍然是根据实际情况,对相应软件过程的评估结果。

2. 软件能力成熟度模型简介

CMM 是对软件组织进化阶段的描述。随着软件组织定义、实施、测量、控制和改进其软件过程,软件组织的能力经过这些阶段会逐步改进。这个能力成熟度模型使软件组织能够较容易地确定其当前过程的成熟度并识别出其软件过程执行中的薄弱环节,确定对软件质量和过程改进最为关键的几个问题,从而形成对其过程的改进策略。软件组织只要关注并认真实施一组有限的关键实践活动,就能稳步地改善全组织的软件过程,使全组织的软件过程能力持续增长。

CMM 将软件过程改进分为 5 个成熟度级别。

(1) 初始级(initial)。软件过程的特点是杂乱无章,有时甚至很混乱几乎没有明确定义的步骤,成功完全依赖个人努力和英雄式的核心人物。

(2) 可重复级(repeatable)。建立了基本的项目管理过程来跟踪成本、进度和机能。有必要的过程准则来重复以前在同类项目中的成功。

(3) 已定义级(defined)。管理和工程的软件过程已经文档化、标准化,并综合成整个软件开发组织的标准软件过程。所有的项目都采用根据实际情况修改后得到的标准软件过程来发展和维护软件。

(4) 已管理级(managed)。制定了软件过程和产品质量的详细量度标准。软件过程的产品质量都被开发组织的成员所理解和控制。

(5) 优化级(optimized)。加强了定量分析,通过来自过程质量的反馈和来自新观念、新技术的反馈使过程能持续不断地改进。

CMM 模型提供了一个框架,将软件过程改进的进化步骤组织成 5 个成熟度等级,为过程不断改进奠定了循序渐进的基础。这 5 个成熟度等级定义了一个有序的尺度,用来测量一个组织的软件过程成熟度和评价其软件过程能力。成熟度等级是已得到确切定义的,向成熟软件组织前进途中的平台。每一个成熟度等级为继续改进过程提供一个基础。每一等级包含一组过程目标,通过实施相应的一组关键过程域达到这一组过程目标。每达到成熟度框架的一个等级,就建立起软件过程的一个较完善的环境,导致组织软件能力成熟度的增长。

基于 CMM 模型的产品包括一些诊断工具,可应用于软件过程评价和软件能力评估小组以确定一个机构的软件过程实力、弱点和风险。最著名的是成熟度调查表。软件过程评价和软件能力评估的方法和培训也依赖于 CMM 模型。

10.2 系统分析基础知识

10.2.1 系统分析概述

1. 系统分析的目的和任务

系统分析的主要任务是对现行系统做进一步的详细调查,将调查所得到的文档资料集中,对组织内部整体管理状况和信息处理过程进行分析,为系统开发提供所需资料,并提交系统方案说明书。系统分析侧重于从业务全过程的角度进行分析,主要内容有:业务和数据的流程是否通畅,是否合理;数据、业务过程和组织管理之间的关系;原系统管理模式改革和新系统管理方法的实现是否具有可行性等。

确定的分析结果包括开发者对于现有组织管理状况的了解,用户对信息系统功能的需求,数据和业务流程,管理功能和管理数据指标体系以及新系统拟改动和新增的管理模型等。

最后,提出信息系统的各种设想和方案,并对所有的设想和方案进行分析、研究、比较、判断和选择,获得一个最优的新系统的逻辑模型,并在用户理解计算机系统的工作流程和处理方式的情况下,将它明确地表达成书面资料——系统分析报告,即系统方案说明书。

2. 系统分析的主要步骤

企业信息系统是一个具有业务复杂性和技术复杂性的大系统,为了使目标系统既能实现当前系统的基本职能,又能改进和提高。系统开发人员首先必须理解并描述出已经实际存在的当前系统,然后进行改进,从而创造出既基于当前系统,又高于当前系统的目标系统,即新系统。

系统分析过程如图 10-6 所示。

- (1) 认识、理解当前的现实环境,获得当前系统的“物理模型”;
- (2) 从当前系统的“物理模型”抽象出当前系统的“逻辑模型”;
- (3) 对当前系统的“逻辑模型”进行分析和优化,建立目标系统的“逻辑模型”;
- (4) 对目标系统的逻辑模型具体化(物理化),建立目标系统的物理模型。

系统分析的目的是把现有系统的物理模型转化为目标系统的物理模型,即图 10-6 中双虚线所描述的路径。而系统分析阶段的结果是得到目标系统的逻辑模型。逻辑模型反映了系统的功能和性质,而物理模型反映的是系统的某一种具体实现方案。

按照图 10-6 所示,可将系统分析阶段的主要工作步骤分为:

- ① 对当前系统进行详细调查,收集数据;

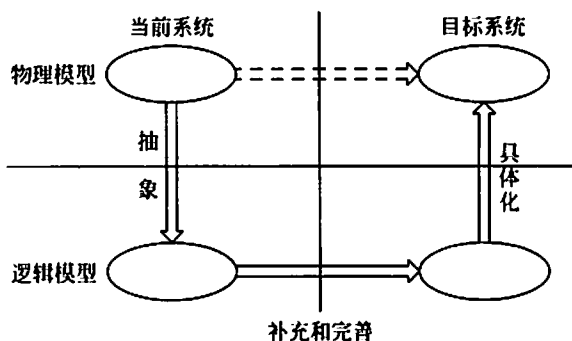


图 10-6 系统分析过程图

- ② 建立当前系统的逻辑模型；
- ③ 对现状进行分析，提出改进意见和新系统应达到的目标；
- ④ 建立新系统的逻辑模型；
- ⑤ 编写系统方案说明书。

10.2.2 系统分析方法

1. 结构化分析方法

结构化分析方法是由美国 Yourdon 公司在 20 世纪 70 年代提出的，其基本思想是将系统分析看成工程项目，有计划、有步骤地进行工作。这是一种应用很广泛的开发方法，适用于分析大型信息系统。结构化分析方法采用“自顶向下，逐层分解”的开发策略。按照这种策略，再复杂的系统也可以有条不紊地进行，只要将复杂的系统适当分层，每层的复杂程度即可降低。

1) 数据流图

数据流图也称数据流程图(data flow diagram, DFD)，是一种便于用户理解和分析系统数据流的图形工具。它摆脱了系统的物理内容，精确地在逻辑上描述系统的功能、输入、输出和数据存储等，是系统逻辑模型的重要组成部分。

(1) DFD 的基本成分。

DFD 的基本成分及其图形表示方法如图 10-7 所示。

- 数据流。数据流由一组固定成分的数据组成，表示数据的流向。值得注意的是：DFD 中描述的是数据流，而不是控制流。除了流向数据存储或从数据存储流出的数据流不必命名外，每个数据流都必须有一个合适的名字，以反映该数据流的含义。
- 加工。加工描述了输入数据流到输出数据流之间的变换，也就是输入数据流经过

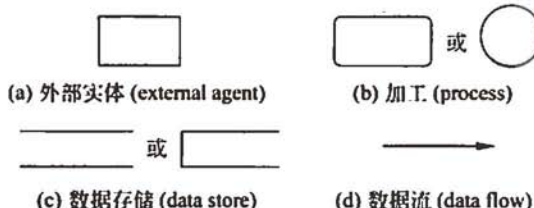


图 10-7 DFD 的基本成分

什么处理后变成了输出数据流。每个加工有一个名字和编号。编号能反映出该加工位于分层 DFD 中的哪个层次和哪张图中,也能够看出它是哪个加工分解出来的子加工。

- 数据存储。数据存储用来表示暂时存储的数据,每个数据存储都有一个名字。
- 外部实体。外部实体是指存在于软件系统之外的人员或组织。它指出系统所需数据的发源地和系统所产生的数据的归宿地。

(2) 分层数据流图的画法。

① 第一步:画系统的输入和输出。把整个软件系统视为一个大的加工,然后根据系统从哪些外部实体接收数据流,以及系统发送数据流到哪些外部实体,就可以画出系统的输入和输出图,这张图称为顶层图。

② 第二步:画系统的内部。将顶层图的加工分解成若干个加工,并用数据流将这些加工连接起来,使得顶层图中的输入数据经过若干个加工处理后转换成顶层图的输出数据流。这张图称为 0 层图。从一个加工画出一张数据流图的过程实际上就是对这个加工的分解。

可以用下述方法来确定加工:在数据流的组成或值发生变化的地方应画一个加工,这个加工的功能就是实现这一变化;也可根据系统的功能确定加工。

确定数据流的方法:用户把若干个数据当作一个单位来处理(这些数据一起到达,一起加工)时,可把这些数据看成一个数据流。

对于一些以后某个时间要使用的数据可以组织成一个数据存储来表示。

③ 第三步:画加工的内部。把每个加工看作一个小系统,把加工的输入输出数据流看成小系统的输入输出数据流。于是可以用画 0 层图同样的方法画出每个加工的 DFD 子图。

④ 第四步:对第三步分解出来的 DFD 子图中的每个加工,重复第三步的分解过程,直至图中尚未分解的加工都足够简单(也就是说这种加工不必再分解)为止。至此得到了一套分层数据流图。

(3) 对图和加工进行编号。

对于一个软件系统,其数据流图可能有许多层,每一层又有许多张图。为了区分不同的加工和不同的 DFD 子图,应该对每张图和每个加工进行编号,以利于管理。

假设分层数据流图里的某张图(记为图 A)中的某个加工可用另一张图(记为图 B)来分解,就称图 A 是图 B 的父图,图 B 是图 A 的子图。在一张图中,有些加工需要进一步分解,有些加工则不必分解。因此,如果父图中有 n 个加工,那么它可以有 $0 \sim n$ 张子图(这些子图位于同一层),但每张子图都只对应于一张父图。

- 顶层图只有一张,图中的加工也只有一个,所以不必编号。
- 0 层图只有一张,图中的加工号可以分别是 0.1,0.2,...或者是 1,2,...
- 子图号就是父图中被分解的加工号。
- 子图中的加工号由图号、圆点和序号组成。

例如,某图中的一个加工号为 2.4,这个加工分解出来的子图号就是 2.4,子图中的加工号分别为 2.4.1,2.4.2,...

(4) 实例。

某考务处理系统有如下功能:

- 对考生送来的报名单进行检查;
- 对合格的报名单进行检查;
- 对阅卷站送来的成绩清单进行检查,并根据考试中心指定的合格标准审定合格者;
- 制作考生通知单(内含成绩合格/不合格标志)送给考生;
- 按地区、年龄、文化程度、职业和考试级别等进行成绩分类统计和试题难度分析,产生统计分析表。

该考务处理系统的分层数据流图如图 10-8 所示。

(5) 应注意的问题

- 适当地为数据流、加工、数据存储及外部实体命名,名字应反映该成分的实际含义,避免使用空洞的名字。
- 画数据流而不要画控制流。
- 一个加工的输出数据流不应与输入数据流同名,即使它们的组成成分相同。
- 允许一个加工有多条数据流流向另一个加工,也允许一个加工有两个相同的输出数据流流向两个不同的加工。
- 保持父图与子图平衡。也就是说,父图中某加工的输入输出数据流必须与它的子图的输入输出数据流在数量和名字上相同。值得注意的是,如果父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流,而子图中组成这些数据

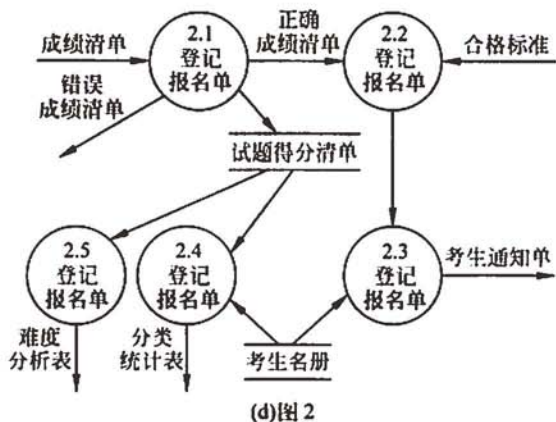
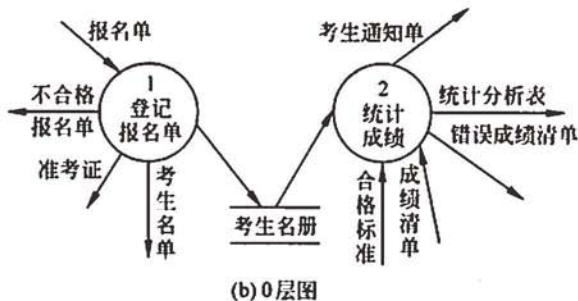
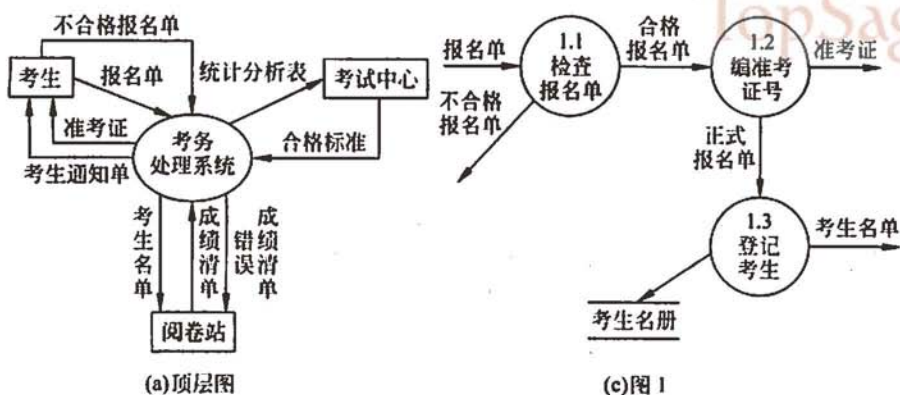


图 10-8 考务处理系统分层数据流图

流的数据项全体正好是父图中的这一个数据流,那么它们仍然算是平衡的。

- 在自顶向下的分解过程中,若一个数据存储首次出现时只与一个加工有关,那么这个数据存储应作为这个加工的内部文件而不必画出。

- 保持数据守恒。也就是说,一个加工的所有输出数据流中的数据必须能从该加工的输入数据流中直接获得,或者是通过该加工能产生的数据。
- 每个加工必须既有输入数据流,又有输出数据流。
- 在整套数据流图中,每个数据存储必须既有读的数据流,又有写的数据流。但在某一张子图中可能只有读没有写,或者只有写没有读。

2) 数据词典(DD)

数据流图描述了系统的分解,但没有对图中各成分进行说明。数据词典就是为数据流图中的每个数据流、文件和加工,以及组成数据流或文件的数据项做出说明。其中对加工的描述称为“小说明”,也可以称为“加工逻辑说明”。

(1) 词典条目。

- 数据流条目。
- 文件条目。
- 数据项条目。

(2) 词典管理。

词典管理主要是把词典条目按照某种格式组织后存储在词典中,并提供排序、查找、统计等功能。如果数据流条目包含了来源和去向,文件条目包含了读文件和写文件,还可以检查数据词典与数据流图的一致性。

3) 描述加工的结构化语言

常用的加工逻辑描述方法有 3 种:结构化语言、判定表和判定树。下面对结构化语言做简单介绍。

用自然语言描述加工逻辑是最方便的,但是自然语言往往不够精确。它可能存在二义性,而且难以用计算机处理。而形式化语言能严格精确地描述事物,且易于计算机处理,但不易被用户理解,因此可以采用结构化语言(如结构化英语)。结构化语言是一种介于自然语言和形式化语言之间的半形式化语言,是自然语言的一个受限子集。

结构化语言没有严格的语法,通常可分为内外两层。外层有严格的语法,如结构化英语的外层可以是 if-then-else、for-do、while-do、set、cope 和 case 等结构,而内层的语法比较灵活,可以接近于自然语言。

例如“折旧策略”的加工逻辑可以用结构化英语描述如下:

```
Depreciation policy
If the Current-Capital-Value is less than $ 1000
Then:
Set Depreciation-Amount to Current-Capital-Value
Set Current-Capital-Value to zero
```

Otherwise:

Set Depreciated-Amount to 10% of Current-Capital-Value

Reduce Current-Capital-Value by 10%

2. 面向对象分析方法

面向对象分析(object-oriented analysis, OOA)的目标是建立待开发软件系统的模型。OOA 模型描述了表示某个特定应用领域中的对象、对象间的结构关系和通信关系,反映了现实世界强加给软件系统的各种规则和约束条件。OOA 模型还规定了对象如何协同工作和完成系统的职责。

20 世纪 80 年代以来相继出现了多种面向对象分析和设计的方法,较为流行的有 Booch 方法、Coad 和 Yourdon 方法、Jacobson 方法等。

统一建模语言(unified modeling language, UML)由于其简单、统一,又能够表达软件设计中的动态和静态信息,目前已经成为可视化建模语言事实上的工业标准。它是一种富有表达力的语言,可以描述开发所需要的各种视图,然后以此为基础装配系统。

UML 基本元素的图形表示如图 10-9 所示。

UML 提供了下列 9 种图:

(1) 类图(class diagram)展现了一组对象、接口、协作和它们之间的关系。在面向对象的建模中最常见的图就是类图。类图给出了系统的静态设计视图,包含主动类的类图给出了系统的静态进程视图。

(2) 对象图(object diagram)展现了一组对象以及它们之间的关系。对象图描述了在类图所建立的事物实例的静态快照。和类图一样,这些图给出了系统的静态设计视图或静态进程视图,但它们是从真实的或原型案例的角度建立的。

(3) 用例图(use case diagram)展现了一组用例、主角(Actor)以及它们之间的关系。用例图给出系统的静态用例视图,用这些图对系统的行为进行组织和建模是非常重要的。

(4) 序列图(sequence diagram)是场景(scenario)的图形化表示,描述了以时间顺序组织的对象之间的交互活动。

(5) 协作图(collaboration diagram)强调收发消息的对象的结构组织。

序列图和协作图都是交互图(interaction diagram)。交互图展现了一种交互,由一组对象和它们之间的关系组成,包括它们之间可能发送的消息。交互图关注系统的动态视图。序列图和协作图是同构的,它们之间可以相互转换。

(6) 状态图(statechart diagram)展现了一个状态机,由状态、转换、事件和活动组成。状态图关注系统的动态视图,对于接口、类和协作的行为建模尤为重要,它强调对象行为的事件顺序。

• 活动图(activity diagram)是一种特殊的状态图。它展现了在系统内从一个活动

到另一个活动的流程。活动图专注于系统的动态视图。它对于系统的功能建模特别重要,并强调对象间的控制流程。

- 构件图(component diagram)展现了一组构件之间的组织和依赖。构件图专注于系统的静态实现视图。它与类图相关,通常把构件映射为一个或多个类、接口或协作。
- 部署图(deployment diagram)展现了运行处理节点及其构件的配置。部署图给出了体系结构的静态实施视图。它与构件图相关,通常一个节点包含一个或多个构件。

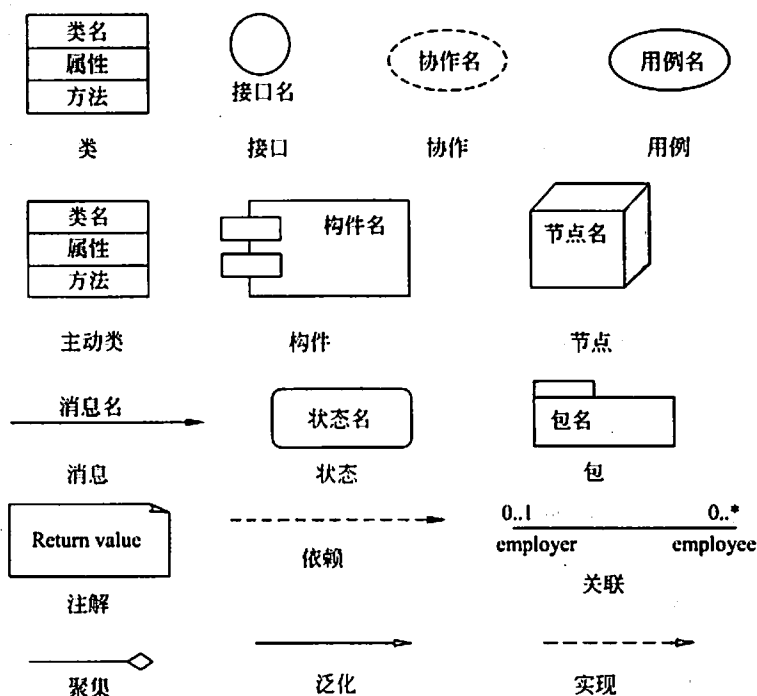


图 10-9 UML 元素的图形表示

10.2.3 系统分析报告

完成整个系统分析阶段的工作后,作为工作成果,应提交一份完整的系统分析报告。系统分析报告一经确认,就成为具有约束力的指导性文件,成为下一阶段系统设计工作的依据和今后验收目标系统的检验标准。系统分析报告形成后必须组织各方面的人员(包括组织的领导、管理人员、专业技术人员、系统分析人员等)一起对已经形成的方案进行论

证,尽可能发现其中的问题和不足。对于有争论的问题要重新核实当初的原始调查资料或进行更深入的调查研究,对于重大的问题甚至可能需要调整或修改系统目标,重新进行系统分析。

在系统分析报告中,数据流图、数据字典和加工说明这3部分是主体,是系统分析报告中必不可少的组成部分。而其他各部分的内容,则应根据所开发的目标系统的规模和性质等具体情况酌情选用,不必生搬硬套。总之,系统分析报告必须简明扼要,抓住本质,反映出目标系统的全貌和开发人员的设想。

系统分析报告主要有以下3个作用:

- (1) 描述了目标系统的逻辑模型,可作为开发人员进行系统设计和实施的基础。
- (2) 作为用户和开发人员之间的协议或合同,为双方的交流和监督提供基础。
- (3) 作为目标系统验收和评价的依据。

因此,系统分析报告是系统开发过程中的一份重要文档,必须完整、一致、精确且简明易懂。

一份完整的系统分析报告应该包括下述内容:

(1) 组织情况概述。

- 对分析对象的基本情况作概括性的描述,包括组织的结构、组织的目标、组织的工作过程和性质、业务功能。
- 系统与外部实体(其他系统或机构)间有哪些物质以及信息的交换关系和联系。
- 参考资料和专门术语说明。

(2) 现行系统概述。

- 现行系统现状调查说明。通过现行系统的组织结构图、数据流图和概况表等,说明现行系统的目标、规模、主要功能、组织机构、业务流程、数据存储和数据流,以及存在的薄弱环节。
- 系统需求说明。用户要求以及现行系统主要存在的问题等。

(3) 系统逻辑模型。

- 新系统拟定的业务流程及业务处理方式。提出明确的功能目标,并与现行系统进行比较和分析,要重点突出计算机处理的优越性。
- 新系统拟定的数据指标体系和分析优化后的数据流程,各个层次的数据流图、数据字典和加工说明,以及计算机系统将完成的工作部分。
- 出错处理要求。
- 其他特性要求。例如系统的输入输出格式、启动和退出等。
- 遗留问题。根据目前条件,暂时不能满足的一些用户要求或设想,并提出今后解决的措施和途径。

- (4) 新系统在各个业务处理环节拟采用的管理方法、算法或模型。
- (5) 与新的系统相配套的管理制度和运行体制的建立。
- (6) 系统设计与实施的初步计划。
 - 工作任务的分解。根据资源及其他条件确定各子系统开发的先后次序,在此基础上分解工作任务,落实到具体组织和个人。
 - 根据系统开发资源与进度估计,制定时间安排计划。
 - 预算。对开发费用的进一步估计。
- (7) 用户领导审批意见。

10.3 系统设计知识

10.3.1 系统设计概述

系统设计是信息系统开发过程中的另一个重要阶段。在这一阶段中,要根据前一阶段系统分析的结果,在已经获得批准的系统分析报告的基础上,进行新系统设计。

系统设计的主要目的就是为系统制定蓝图,在各种技术和实施方法中权衡利弊,精心设计,合理使用各种资源,最终勾画出新系统的详细设计方案。

系统设计阶段的主要依据是系统分析报告和开发者的经验。系统设计的主要内容包括新系统总体结构设计、代码设计、输出设计、输入设计、处理过程设计、数据存储设计、用户界面设计和安全控制设计等。

系统设计的基本任务大体上可以分为两个步骤。

① 把总任务分解为许多基本的、具体的任务。把这些具体任务合理地组织起来构成总任务,称为总体结构设计(architecture design),又称为概要结构设计(preliminary design)。其基本任务是:将系统划分为模块,决定每个模块的功能,决定模块的调用关系,决定模块的接口及界面,即模块间信息的传递。

总体结构设计是系统开发过程中非常关键的一步。系统的质量及一些整体特性基本上是这一步决定的。系统越大,总体结构设计的影响越大。各个局部都很好,组合起来就一定好的想法是不实际的,因为部分最优,并不能保证总体最优。

② 为各个具体任务选择适当的技术手段和处理方法,即详细设计。内容包括代码设计、输出设计、输入设计、处理过程设计、数据存储设计、用户界面设计和安全控制设计。

系统设计的结果是一系列系统设计文件,这些文件是具体实现一个信息系统(包括硬件设备和编制软件程序)的重要基础。

10.3.2 系统总体结构设计

系统总体结构设计要根据系统分析的要求和组织的实际情况,对新系统的总体结构形式和可利用的资源进行宏观设计,这是一种宏观、总体上的设计和规划。下面介绍系统总体设计的主要内容。

1. 系统总体结构设计原则

为保证总体结构设计的顺利完成,主要应遵循以下几条原则:

(1) 分解—协调原则。整个系统是一个整体,具有整体目标和功能。但这些目标和功能的实现又是由相互联系的各个组成部分共同工作的结果。解决复杂问题的一个很重要的原则就是把它分解成多个小问题分别处理,在处理过程中根据系统总体要求协调各部门的关系。

(2) 自顶向下的原则。首先抓住系统总的功能和目的,然后逐层分解,即先确定上层模块的功能,再确定下层模块的功能。

(3) 信息隐蔽、抽象的原则。上层模块只规定下层模块做什么和所属模块间的协调关系,但不规定怎么做,以保证各模块的相对独立性和内部结构的合理性,使得模块与模块之间层次分明,易于理解,易于实施,易于维护。

(4) 一致性原则。要保证整个软件设计过程中具有统一的规范、统一的标准和统一的文件模式等。

(5) 明确性原则。每个模块必须功能明确,接口明确,消除多重功能和无用接口。

(6) 模块之间的耦合尽可能小,模块内部组合要尽可能紧凑。

(7) 模块的扇入系数和扇出系数要合理。一个模块直接调用其他模块的个数称为模块的扇出系数;反之,一个模块被其他模块调用时,直接调用的它的模块个数称为模块的扇入系数。模块的扇入、扇出系数必须适当。经验表明,一个设计得好的系统的平均扇入、扇出系数通常是3或4,一般不应超过7,否则会引起出错概率的增大。但菜单调用型模块的扇入与扇出系数可以大一些,公用模块扇入系数可以大一些。

(8) 模块的规模适当。过大的模块常常是因为系统分解得不充分,其内部可能包含了若干部分的功能,因此有必要进一步把原有的模块分解成若干功能尽可能单一的模块。但分解也必须适度,因为过小的模块有可能降低模块的独立性,造成系统接口的复杂性。一条有益的经验是,一个模块的规模最好使程序清单限制在1~2页纸内,这样的模块易于编制、维护和修改。

2. 子系统划分

1) 子系统划分的原则

为了便于以后的系统开发和系统运行,子系统的划分应遵循以下几点原则:

- 子系统要具有相对独立性。
- 子系统之间数据的依赖性尽量小。
- 子系统划分的结果应使数据冗余较小。
- 子系统的设置应考虑今后管理发展的需要。
- 子系统的划分应便于系统分阶段实现。
- 子系统的划分应考虑到各类资源的充分利用。

2) 子系统结构设计

子系统结构设计任务是确定划分后的子系统模块结构,并画出模块结构图。这个过程中必须考虑以下几个问题:

- 每个子系统如何划分成多个模块。
- 如何确定子系统之间、模块之间传送的数据及其调用关系。
- 如何评价并改进模块结构的质量。
- 如何从数据流图导出模块结构图。

3. 系统模块结构设计

模块设计是系统设计的一个重要阶段。在模块设计阶段,系统划分成各个基础部分——模块,确定系统的总体结构。总体结构与各个分层模块结构的关系是程序实施的重要依据。模块结构采用模块结构图来表示。模块结构图是采用 HIPO(分层输入—处理—输出)图形式绘制而成的框图。

1) 模块的概念

模块是组成系统的基本单位,它的特点是可以组合、分解和更换。系统中任何一个处理功能都可以看成是一个模块。根据模块功能具体化程度的不同,可以分为逻辑模块和物理模块。在系统逻辑模型中定义的处理功能可视为逻辑模块。物理模块是逻辑模块的具体化,可以是一个计算机程序、子程序或干条程序语句,也可以是人工过程的某项具体工作。

一个模块应具备以下 4 个要素。

- 输入和输出: 模块的输入来源和输出去向都是同一个调用者,即一个模块从调用者那里取得输入,进行加工后再把输出返回给调用者。
- 处理功能: 指模块把输入转换成输出所做的工作。
- 内部数据: 指仅供该模块本身引用的数据。
- 程序代码: 指用来实现模块功能的程序。

前两个要素是模块的外部特性,反映了模块的外貌,后两个要素是模块的内部特性。在结构化设计中,主要考虑的是模块的外部特性,对其内部特性只做必要了解,具体的实现将在系统实施阶段完成。

2) 模块结构图

为了保证系统设计工作的顺利进行,结构设计应遵循如下原则:

- 模块的内部凝聚性要强,模块之间的联系要少,即模块具有较强的独立性。
- 模块之间的连接只能存在上下级之间的调用关系,不能有同级之间的横向联系。
- 整个系统呈树状结构,不允许网状结构或交叉调用关系出现。
- 所有模块(包括后继 IPO 图)都必须严格地分类编码并建立归档文件。

模块结构图主要关心的是模块的外部属性,即上下级模块、同级模块之间的数据传递和调用关系,与模块的内部无关。

模块结构图是结构化设计中描述系统结构的图形工具。作为一种文档,它必须严格地定义模块的名字、功能和接口,同时还应当在模块结构图上反映出结构化设计思想。模块结构图由模块、调用、数据、控制和转接等 5 种基本符号组成,如图 10-10 所示。



图 10-10 模块结构图的基本符号

- 模块。这里所说的模块通常是指用一个名字就可以调用的一段程序语句。长方形中间标上能反映模块处理功能的模块名字。
- 调用。箭头总是由调用模块指向被调用模块,但应该理解为被调用模块执行后又返回到调用模块。

如果一个模块是否调用一个从属模块,取决于调用模块内部的判断条件,则该调用模块间的判断采用菱形符号表示。如果一个模块通过其内部循环的功能来循环调用一个或多个从属模块,则该调用称为循环调用,用弧形箭头表示。判断调用和循环调用的表示方法如图 10-11 所示。



图 10-11 模块调用示例

- 数据。当一个模块调用另一个模块时,调用模块可以把数据传送到被调用模块供其处理,而被调用模块又可以将处理的结构送回到被调用模块。在模块之间传送的数据,使用与调用箭头平行的带空心圆的箭头表示,并在旁边标上数据名。图 10-12(a)表示模块 A 调用模块 B 时,A 将数据 x、y 传送给 B,B 将处理结果数据 z

返回给 A。

- 控制信息。模块间有时还必须传送某些控制信息。例如,数据输入完成后给出的结束标志,文件读到末尾时所产生的文件结束标志等。控制信息与数据的主要区别是前者只反映数据的某种状态,不必进行处理。图 10-12(b)中“无此职工”就是表示送来的职工号有误的控制信息。

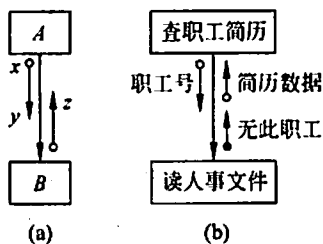


图 10-12 模块间的数据传递

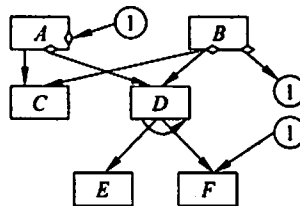


图 10-13 转接符号的使用

- 转接符号。当模块结构图在一张纸上画不下,需要转接到另一张纸上,或者为了避免图上线条交叉时,都可以使用转接符号(圆圈内加上标号)如图 10-13 所示。

4. 数据存储设计

信息系统的主要任务是通过大量的数据获得管理所需要的信息,这就必须存储和管理大量的数据。因此,建立一个良好的数据组织结构和数据库,使整个系统都可以迅速、方便、准确地调用和管理所需的数据,是衡量信息系统开发工作好坏的主要指标之一。

数据结构组织、数据库或文件设计,就是要根据数据的不同用途、使用要求、统计渠道、安全保密性等,来解决数据的整体组织形式、表或文件的形式,以及决定数据的结构、类别、载体、组织方式、保密级别等一系列问题。

一个好的数据结构和数据库应该充分反映物流发展变化的状况,充分满足组织的各级管理要求。同时还应该使得后继系统开发工作方便快捷,系统开销(如占用空间、网络传输频度、磁盘或光盘读写次数等)小,易于管理和维护。

在确立了数据的整体结构之后,剩下的就是要确定数据资源分布和安全保密属性了。其中数据资源的分布是针对分布式数据库系统而言的,而安全保密属性的定义则是针对某些特殊信息,如财务数据等而言的。

- 数据资源分布:如果所规划和设计的系统是在网络环境之下,那么数据库设计必须考虑整个数据资源在网络各节点(包括网络服务器)上的分配问题。
- 数据的安全保密:一般数据库软件都提供定义数据安全保密性的基本功能。系统所提供的安全保密功能一般有 8 个等级(0~7 级),4 种不同方式(只读、只写、删除和修改),而且允许用户利用这 8 个等级的 4 种方式对每一个表自由地进行

定义。

10.3.3 系统详细设计

1. 代码设计

代码是用来表征客观事物的一组有序的符号,以便于计算机和人工识别与处理。代码的类指代码符号的表示形式,一般有数字型、字母型以及数字字母混合型等。3种类型的代码各有所长,应根据使用者的要求、信息量的多少、信息交换的频度以及使用者的习惯等方面综合考虑。

代码设计应该遵循以下基本原则:

- 惟一性。一个对象可能有多个名称,也可按不同的方式对它进行描述。但在一个编码体系中,一个对象只能赋予惟一的代码。最简单的就是职工编号和学生的学号等。
- 合理性。代码结构与响应的分类体系相对应。
- 可扩充性。应留有充分的余地,以备将来不断扩充的需要。
- 简单性。结构尽可能简单,以减少各种差错。
- 适用性。代码尽可能反映对象的特点,以助记忆,便于填写。
- 规范性。国家有关编码标准是代码设计的重要依据,已有的标准必须遵循。在一个代码体系中,代码结构、类型和编写格式必须统一。
- 系统性。有一定的分组规则,从而在整个系统中具有通用性。例如,在会计领域中,一级会计科目有国家财政部分类标准,二级会计科目有各部委或行业协会统一的分类标准,并且这个分类还必须参照一二级科目的规律进行。

代码设计可以按照以下步骤进行:

- ① 确定代码对象。
- ② 考察是否已有标准代码。如果国家或者某个部门已对某些事物规定了标准代码,那么应遵循这些标准代码。
- ③ 根据代码的使用范围、使用时间和实际情况选择代码的种类与类型。
- ④ 考虑检错功能。
- ⑤ 编写代码表。代码编好后,要编制代码表,做详细说明,通知有关部门安排培训,以便正确使用。

目前常用的编码有顺序码、数字码、字符码和混合码;限于篇幅,这里不做详细论述。

2. 输出设计

从系统开发的角度看,输出决定输入,即输入信息只有根据输出要求才能确定。输出设计包括以下几方面的内容。

- 确定输出内容。输出内容的设计首先要确定用户在使用信息方面的要求。根据用户要求,设计输出信息的内容,包括对信息形式(表格、图形、文字)、输出项目、数据结构、数据类型、位数及取值范围、数据的生成途径、完整性及一致性的考虑等。
- 选择输出设备与介质。常用的输出设备有显示终端、打印机、磁带机、绘图仪、缩微胶卷输出器以及多媒体设备。输出介质有纸张、磁带、磁盘、缩微胶卷、光盘和多媒体介质等。这些设备和介质各有特点,应根据用户对输出信息的要求,结合现有设备和资金条件选择。
- 确定输出格式。输出格式要满足使用者的要求和习惯,做到格式清晰、美观、易于阅读和理解。

最终输出方式常用的只有两种:一种是报表输出,另一种是图形输出。采用哪种输出形式为宜,应根据系统分析和业务管理的要求而定。一般来说,对于基层和具体事务的管理者,应用报表方式给出详细的记录数据为宜;而对于高层领导或宏观、综合管理部门,则应该使用图形方式以显示综合数据或发展趋势等信息。

3. 输入设计

输入设计的目的是保证向系统输入正确的数据。在此前提下,做到输入方法简单、迅速、经济、方便。为此,输入设计应遵循以下原则:

- 最小量原则。即保证满足处理要求的前提下使输入量最小。输入量越少,出错机会越小,花费时间越少,数据一致性越好。
- 简单性原则。输入的准备、输入过程应尽量容易,以减少错误的发生。
- 早检验原则。对输入数据的检验应尽量接近源数据发生点,使错误能及时得到改正。
- 少转换原则。输入数据尽量使用其处理所需的形式记录,以免数据转换介质时发生错误。

输入设计的内容包括:

- 确定输入数据内容。包括确定输入数据项名称、数据内容、精度和数值范围。
- 输入方式设计。主要是根据总体设计和数据库设计的要求来确定数据输入的具体形式。常用的输入方式有键盘输入,模/数、数/模输入,网络数据传送,磁/光盘读入等。通常在设计新系统的输入方式时,应尽量利用已有的设备和资源,避免大批量的数据通过键盘输入。
- 输入格式设计。实际设计数据输入(特别是大批量的数据统计报表输入)时,常常遇到统计报表(文件)结构与数据库文件结构不完全一致的情况,如有可能,应尽量统一统计报表或数据库关系表二者的结构,使其一致,以减少输入格式设计的

难度。现在还可采用智能输入方式,由计算机自动将输入送至不同表格。

- 校对方式设计。特别是针对数字、金额数等字段,没有适当的校对措施作保证是很危险的。所以对一些重要的报表,输入设计一定要考虑适当的校对措施,以减少出错的可能性。但应指出的是没有绝对保证不出错的校对方式。

常用的校对方式有人工校对、二次键入校对(同一批数据两次键入)和数据平衡校对。

4. 处理过程设计

总体结构设计将系统分解成许多模块,并决定每个模块的外部特征,即功能和界面。计算机处理过程的设计则要确定每个模块的内部特征,即内部的执行过程,包括局部的数据组织、控制流、每一步的具体加工要求及种种实施细节。通过这样的设计,为编写程序制定一个周密的计划。

处理过程设计的关键是用一种合适的表达方法来描述每个模块的执行过程。这种表示方法应该简明、精确,并由此能直接导出用编程语言表示的程序。常用的描述方式有图形、语言和表格 3 类,如传统的框图、各种程序语言和判定表等。

1) 程序流程图

流程图(flow chart)即程序框图,是历史最久、流行最广的一种图形表示方法。程序流程图包括 3 种基本成分:加工步骤,用方框表示;逻辑条件,用菱形表示;控制流,用箭头表示。

图形表示的优点是直观、形象,容易理解。但从结构化程序设计的角度看,流程图不是理想的表达工具。缺点之一是表示控制的箭头过于灵活。使用得当,流程图简单易懂;使用不当,流程图可能非常难懂,而且无法维护。流程图的另一个缺点是只描述执行过程而不能描述有关数据。

2) 盒图(NS 图)

盒图是结构化程序设计出现之后,为支持这种设计方法而产生的一种描述工具。在 NS 图中,每个处理步骤用一个盒子表示。盒子可以嵌套。盒子只能从上头进入,从下头走出,除此之外别无其他出入口,所以盒图限制了随意的控制转移,保证了程序的良好结构。

3) 形式语言

形式语言是用来描述模块具体算法的非正式的、比较灵活的语言。其外层语法是确定的,而内层语法不确定。外层语法用类似一般编程语言的保留字描述控制结构,所以是确定的。内层语法故意不确定,可以按系统的具体情况和不同层次灵活选用,实际上可用自然语言来描述具体操作。

可以看出形式语言同结构性语言的想法是一致的。形式语言的优点是接近自然语言(英语),所以易于理解;其次,它可以作为注释嵌套在程序中成为内部文档,提高程序的自

我描述性;第三,因为是形式语言,易于被计算机处理,可用行编辑程序或字处理系统对形式语言进行编辑修改。

4) 决策树

如果一个加工决策或判断的步骤较多,则使用形式语言时,语句的嵌套层次也较多,不便于基本加工的逻辑功能的清晰描述。决策树是一种图形工具,适合于描述加工中具有多个策略且每个策略和若干条件有关的逻辑功能。

5) 决策表

在基本加工中,如果判断的条件较多,各个条件又相互组合,相应的决策方案也较多,可用决策树来描述。如果树的结构比较复杂,图中各项注释比较繁琐时,可使用决策表。决策表也是一种图形工具,它可以将比较复杂的决策问题简洁、明确和一目了然地描述出来。

5. 用户界面设计

用户界面是系统与用户之间的接口,也是控制和选择信息输入输出的主要途径。用户界面设计应坚持友好、简便、实用、易于操作的原则。例如,在设计菜单时应尽量避免菜单嵌套层次过多,每选择一次还须确认一下的设计方式。又如,在设计大批量数据输入屏幕界面时,应避免颜色过于鲜艳和多变。

界面设计包括菜单方式、会话方式、操作提示方式以及操作权限管理方式等。

1) 菜单方式

菜单是信息系统功能选择操作的最常用方式。按目前软件所提供的菜单设计工具,菜单的形式可以是下拉式、弹出式的,也可以是按钮选择方式的。

2) 会话管理方式

在所有的用户界面中,几乎毫无例外都会遇到人机会话问题。最为常见的有:当用户操作错误时,系统向用户发出提示和警告性信息;当系统执行用户操作命令遇到多种可能时,系统会要求用户进一步说明;系统定量分析的结果通过屏幕向用户发出控制型的信息等。这类会话的处理方式是让系统开发人员根据实际系统操作过程将会话语句写在程序中。

一般会话系统是要面向企业领导的,会话系统设计必须满足会话的基本要求,如画面清晰,直观形象,明了简洁,具有容错和纠错能力,提供信息汉化、图形化、表格化等功能。

因此,会话设计的重点是解决会话方式、容错能力和系统的模块结构。

在语音会话方式还没有广泛使用的今天,会话的基本工具是键盘、屏幕和打印机,常用的方式是回答式、菜单式、表格式和图形式。

3) 提示方式与权限管理

为了操作和使用方便,在设计系统时,常常把操作提示和要点同时显示在屏幕的旁边,以使用户操作方便,这是当前比较流行的用户界面设计方式。另一种操作提示设计方式则是将整个系统操作说明书全送入到系统文件中,并设置系统运行状态指针。当系统运行时,指针随着系统运行状态来改变,当用户按“帮助”键时,系统则立刻根据当前指针调出相应的操作说明。

与操作方式有关的另一个内容就是对数据操作权限的管理。权限管理一般都是通过入网口令和建网时定义该节点级别,这两点相结合来实现的。

6. 安全控制设计

从数据环境和数据处理两方面看,影响系统安全的因素如下所述。

- 环境性因素,是指管理机构的组织、硬件、系统软件、系统开发和自然环境等方面的因素。例如:组织方面职责不分,没有监督机构等;软硬件方面的硬件失灵、系统软件失灵,逻辑线路错误等;系统开发方面没有按科学的方法开发系统和设计程序,系统未经测试和调试等;自然环境方面的火灾、水灾、风灾、地震等;安全管理方面资源的接触是随意的,无必要的限制等。
- 数据处理因素,是指数据处理行为引起的各种情况。例如:输入环节录入错误信息,使用无效代码,击错功能键,丢失数据,重复输入,没有将数据存盘等;处理环节使用了错误的程序,使用了错误的文件,处理不及时,丢失数据文件和程序等;输出环节的发送报表错误或未及时发送,报告中的错误未加更正等。

要进行系统的安全控制,应从影响系统安全的这两方面因素入手,有的放矢,相应地进行环境和数据处理两方面的有效控制,以保证系统安全有效地运行。

7. 系统设计说明书

系统设计阶段的最终结果是系统设计报告。系统设计报告是下一步系统实施的基础。

从系统调查、系统分析到系统设计是信息系统开发的主要工作,这3个阶段的工作量几乎占到了总开发量的70%,而且这3个阶段所用的工作图表较多,涉及面广,较为复杂。

一份完整的系统设计说明书应该包含以下内容。

1. 引言

- (1) 背景。
- (2) 摘要。
- (3) 工作条件/限制。
- (4) 参考和引用资料。
- (5) 专门术语定义。

2. 系统总体技术方案

- (1) 模块设计。
- (2) 代码设计。
- (3) 输入设计。
- (4) 输出设计。
- (5) 数据库设计说明。
- (6) 模型库及方法库设计。
- (7) 网络设计。
- (8) 安全保密设计。
- (9) 实施方案说明书。

10.4 系统实施知识

10.4.1 系统实施概述

1. 系统实施的目的和任务

系统实施是新系统开发工作的最后一个阶段。所谓实施指的是将系统设计阶段的结果在计算机上实现,将原来纸面上的、类似于设计图式的新系统方案转换成可执行的应用软件系统。系统实施阶段的主要任务是:

- 按总体设计方案购置和安装计算机网络系统。硬件设备包括计算机主机、输入输出设备、存储设备、辅助设备(稳压电源、空调设备等)、通信设备。购置、安装和调试这些设备要花费大量的人力、物力,并且要持续相当长的时间。
- 软件准备。软件准备包括系统软件、数据库管理系统以及一些应用程序。有些软件需要购买,有些需要组织人力编写。编写程序是系统实施阶段的重要任务之一。
- 人力培训。主要指用户的培训,包括主管人员和业务人员。这些人多数来自现行系统,精通业务,但往往缺乏计算机知识。为了保证系统调试和运行顺利,应根据他们的基础,提前进行培训,使他们适应并逐步熟悉新的操作方法。
- 数据准备。数据的收集、整理、录入是一项既繁重、劳动量又大的工作。而没有一定基础数据的准备,系统调试就不可能很好地进行。一般来说,确定数据库模型之后,就应进行数据的整理、录入。这样既分散了工作量,又可以为系统调试提供真实的数据。
- 试运行。

2. 系统实施的步骤

系统开发工作沿着信息系统的生命周期逐渐推进,经过详细设计阶段后,便进入系统实施阶段。系统实施的步骤如下所示:

① 按总体设计方案购置和安装计算机网络系统。购置和安装硬件是比较简单的事情,只需要按总体设计的要求和可行性报告中财力资源的分析,选择好价格性能比高的设备,通知厂家按要求供货并安装即可。

② 建立数据库系统。如果前面数据与数据流程分析以及数据库设计工作进行得比较规范,而且开发者又对数据库技术比较熟悉的话,按照数据库设计的要求,只需要 1~2 人一天即可建立起一个大型数据库结构。

③ 程序设计。

④ 收集有关数据并进行录入工作,然后进行系统测试。

⑤ 人员培训、系统转换和试运行。

10.4.2 程序设计

程序设计的主要依据是系统设计阶段的 HIPO 图以及数据库结构和编码设计。

1. 程序设计方法

目前程序设计大多按照结构化方法、原型方法、面向对象的方法进行。

编程的目的是为了实现开发者在系统分析和系统设计中提出的管理方法和处理构想。所以在编程和实现中,建议应尽量借用已有的程序和开发工具,尽快尽好地建立系统,而不要在具体的编程和调试工作中花费过多的精力和时间。

- 结构化程序设计方法。若遇到某些开发过程不规范,模块划分不细,或者是因特殊业务处理的需要模块程序量较大时,结构化程序设计方法是一种非常有效的方法。结构化的程序设计方法主要强调 3 点:模块内部程序各部分要按自顶向下的结构划分;各程序部分应按功能组合;各程序之间的联系尽量通过调用子程序 (CALL-RETURN) 来实现,不用或少用 GOTO 方式。
- 快速原型式的程序开发方法。具体实施方法是首先将 HIPO 图中带有普遍性的功能模块集中,如菜单模块、报表模块、查询模块以及统计分析和图像模块等,这些模块几乎是每个子系统必不可少的;然后再去寻找有无相应且可用的软件工具,如果没有则可以考虑开发一个能够适合各子系统情况的通用模块;最后用这些工具生成这些程序模块原型。如果 HIPO 图中有一些特定的处理功能和模块,而这些功能和模块又是利用现有工具不可能生成的,则应考虑编制一段程序加进去,利用现有的工具和原型方法可以很快地开发出所要的程序。
- 面向对象程序设计方法。面向对象程序设计方法一般应与 OOD 所设计的内容相

对应。它是一个简单直接的映射过程。即将 OOD 中所定义的模式直接用面向对象程序(OOP)如 C++、Smalltalk、Visual C 等来取代即可。例如,用 C++ 中的对象类型来取代 OOD 模式中的类-对象,用 C++ 中的函数和计算功能来取代 OOD 模式中的处理功能等。在系统实现阶段,OOP 的优势是巨大的,是其他方法所无法比拟的。

2. 程序设计基本模块

一个信息系统的应用软件由很多程序模块组成,这些程序模块可以归纳为几种基本类型,其结构如图 10-14 所示。

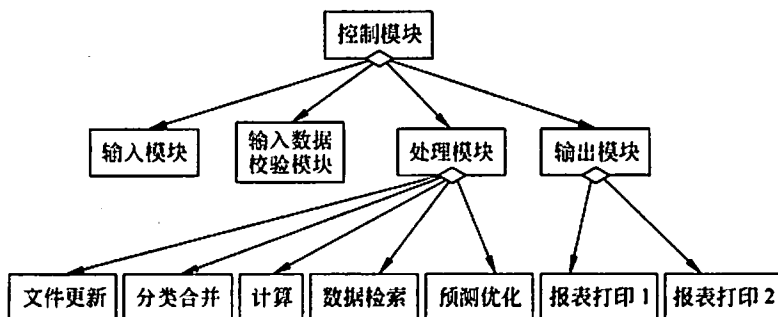


图 10-14 基本程序模块图

(1) 控制模块。控制模块包括主控制模块和各级控制模块。控制模块的主要功能是根据用户要求的信息,由用户确定处理顺序,然后把控制转向各处理模块的入口。

(2) 输入模块。主要用来输入数据,输入方式主要有键盘输入和磁(光)盘输入两种。

(3) 输入数据校验模块。该模块对已经输入计算机中的数据进行校验,以保证原始数据的正确性。校验的方法通常有重复输入校验和程序校验两种。

(4) 输出模块。输出模块用来将计算机的运行结果通过屏幕、打印机或磁盘以及磁带等设备输出给用户。在信息系统中,一般都有大量的表格、图表需要输出,因此输出模块的质量直接关系到整个系统的性能。

(5) 处理模块。根据信息系统的不同应用要求,有不同的处理功能,通常有以下几种:

- 文件更新模块。当系统的应用数据发生变化时,需要修改数据文件。例如,增加新的记录、修改数据项或记录、删除某些不需要的记录等。一般来说,文件更新模块应该具有下述功能:对记录中关键字的控制功能,以便通过关键字查找相应记录;控制总记录数的功能,以便控制追加、插入记录的位置;具有记录地址或字节位置的控制功能,以便确定修改数据的位置,控制插入或者追加数据的位置。

- 分类合并模块。分类合并模块的主要功能是对已经建立的文件,按某关键字进行分类合并。例如,在材料核算系统中耗材要按照材料类型合并处理。分类合并程序应该具有控制记录总数、字符串比较、排序、统计和计数等功能。
- 计算模块。该模块的主要功能是进行计算处理,包括同类记录中各数据项的运算。
- 数据检索模块。该模块的主要功能是为用户提供查询的有关信息,包括输入查询要求和输出特定的查询结果。它是管理信息系统的人机接口,对于人机交互的友好程度以及查询响应时间等均有较高要求。
- 预测或优化模块。该模块的主要功能是使用预测或优化数学模型,利用信息系统提供的有关数据,进行计算和分析并输出结果,用来辅助企业或部门的管理人员进行决策。例如库存管理中的 ABC 分类、最佳订货量计算和财务管理中的资金分析等。

3. 程序设计语言的选择

每种程序设计语言都有自己的特点,为一个特定的开发项目选择编程语言时通常应考虑下列一些因素:应用领域、算法和计算的复杂性、软件运行的环境(包括可使用的编译程序)、用户需求(特别是性能需求)、数据结构的复杂性以及开发人员的水平等。

从应用领域来看。COBOL 语言适用于商业领域的应用,FORTRAN 语言适合于科学和工程计算的应用,PROLOG 和 LISP 适合于人工智能的应用,对于一些采用面向对象方法的应用系统通常可选用 C++ 或 Java。有些程序设计语言可应用于多种应用领域。例如 C 语言,它原先是为辅助开发 UNIX 操作系统而设计的,主要用于开发系统软件,现在已经广泛应用于其他领域。

开发和维护高级语言程序要比开发和维护低级语言程序容易得多,但高级语言程序经过编译后所产生的目标程序要比完成相同功能的低级语言程序长得多,也就是说前者的功效要比后者的功效低。因此某些对运行时间或存储空间有过高要求的项目,或者在不能提供高级语言编译程序的计算机上开发程序,往往要使用低级语言。在某些大型系统中,为了提高系统的运行效率,有时也对系统中在执行时间上起关键作用的模块局部使用低级语言。

当然,所选择的语言必须有可使用的编译程序,如果有多种语言都适合于某项目的开发时,也可以考虑选择开发人员比较熟悉的一种。

10.4.3 系统测试与调试

1. 系统测试的意义及目的

系统测试是为了发现错误而执行程序的过程,成功的测试是发现了至今尚未发现的

错误的测试。

测试的目的就是希望能以最少的人力和时间发现潜在的各种错误和缺陷。应根据开发各阶段的需求、设计等文档或程序的内部结构精心设计测试实例,并利用这些实例来运行程序,以便发现错误。信息系统测试应包括软件测试、硬件测试和网络测试。硬件测试、网络测试可以根据具体的性能指标来进行,此处所说的测试更多的是指软件测试。

系统测试是保证系统质量和可靠性的关键步骤,是对系统开发过程中的系统分析、系统设计和实施的最后复查。根据测试的概念和目的,在进行信息系统测试时应遵循以下基本原则:

- 应尽早并不断地进行测试。
- 测试工作应该避免由原开发软件的人或小组承担,由专门人员来承担会更客观,更有效。
- 设计测试方案的时候,不仅要确定输入数据,而且要根据系统功能确定预期的输出结果。将实际输出结果与预期结果进行比较就能发现测试对象是否正确。
- 在设计测试实例时,不仅要设计有效合理的输入条件,也要包含不合理、失效的输入条件。
- 在测试程序时,不仅要检验程序是否做了该做的事,还要检验程序是否做了不该做的事。多余的工作会带来副作用,影响程序的效率,有时会带来潜在的危害或错误。
- 严格按照测试计划来进行,避免测试的随意性。测试计划应包括测试内容、进度安排、人员安排、测试环境、测试工具和测试资料等。严格按照测试计划进行测试可以保证进度,使各方面都得以协调进行。
- 妥善保存测试计划和测试例子,将其作为软件文档的组成部分,为维护提供方便。
- 测试例子都是精心设计出来的,可以为重新测试或追加测试提供方便。

2. 测试过程

测试是开发过程中一个独立且非常重要的阶段,测试过程基本上与开发过程平行进行。

一个规范化的测试过程通常包括以下基本的测试活动。

(1) 拟定测试计划。在制定测试计划时,要充分考虑整个项目的开发时间和开发进度以及一些人为因素和客观条件等,使得测试计划是可行的。测试计划的内容主要有:测试的内容、进度安排、测试所需的环境和条件、测试培训安排等。

(2) 编制测试大纲。测试大纲是测试的依据,它明确详尽地规定了在测试中针对系统的每一项功能或特性所必须完成的基本测试项目和完成测试的标准。

(3) 根据测试大纲设计和生成测试例子。在设计测试例子的时候,可综合利用前面

介绍的测试例子和设计技术,产生测试设计说明文档,其内容主要有被测项目、输入数据、测试过程、预期输出结果等。

(4) 实施测试。测试的实施阶段是由一系列的测试周期组成的。在每个测试周期中,测试人员和开发人员将依据预先编制好的测试大纲和准备好的测试例子,对被测软件或设备进行完整的测试。

(5) 生成测试报告。测试完成后,要形成相应的测试报告,主要对测试进行概要说明,列出测试的结论,指出缺陷和错误。另外,给出一些建议,如可采用的修改方法,各项修改预计的工作量及修改的负责人员。

3. 测试策略和测试方法

软件测试方法分人工测试和机器测试。

1) 人工测试

人工测试指的是采用人工方式进行测试,目的是通过对程序静态结构的检查,找出编译时不能发现的错误。经验表明,组织良好的人工测试可以发现程序中 30%~70% 的编码和逻辑设计错误。

人工测试又称为代码审查,其内容包括检查代码和设计是否一致,检查代码逻辑表达是否正确和完整,检查代码结构是否合理等。主要有 3 种方法。

- 个人复查:指程序员本人对程序进行检查。由于心理上的原因和思维惯性的影响,对自己的错误一般不容易发现,对功能理解的错误更不可能纠正。因此,这种方法主要针对小规模程序,效率不高。
- 抽查:通常由 3~5 人组成测试小组,测试人员应是没有参加该项目开发的有经验的程序设计人员。在抽查之前,应先阅读相关的软件资料和源程序,然后由测试人员扮演计算机的角色,将一批有代表性的测试数据沿程序的逻辑走一遍,监视程序的执行情况。人工检测程序很慢,只能选择少量简单的例子。
- 会审。测试人员的构成与抽查类似。在会审之前,测试人员应该充分阅读相关资料,比如系统分析说明书、系统设计说明书、源程序等。测试人员应尽可能多地列出典型错误。在会审时,由编程人员逐句讲解程序,测试人员逐个审查、提问。通过这种方式,往往可能使编程人员发现自己以前没有意识到的错误,使问题暴露。会审后,要将发现的问题登记、分析、归类。

代码复审应该在被测软件编译成功之后。编译都不通过的软件,当然谈不上复审。在复审期间,应保证有足够的时间,让测试小组对问题进行充分的讨论,这样才能有效地提高测试效率,避免出错。

2) 机器测试

机器测试是把设计好的测试例子作用于被测程序,比较测试结果和预期结果是否一

致。如果不一致,就说明可能存在错误。机器测试只能发现错误的症状,无法对问题进行定位。

机器测试分为黑盒测试和白盒测试两种。

(1) 黑盒测试也称为功能测试。将软件看成黑盒子,在完全不考虑软件内部结构和特性的情况下,测试软件的外部特性。进行黑盒测试主要是为了发现以下几类错误:

- 是否有错误的功能或遗漏的功能?
- 界面是否有误? 输入是否正确接收? 输出是否正确?
- 是否有数据结构或外部数据库访问错误?
- 性能是否能够接受?
- 是否有初始化或终止性错误?

(2) 白盒测试也称为结构测试。将软件看成透明的白盒。根据程序的内部结构和逻辑来设计测试例子,对程序的路径和过程进行测试,检查是否满足设计的要求。其原则是:

- 程序模块中的所有独立路径至少执行一次。
- 在所有的逻辑判断中,取“真”和取“假”的两种情况至少都要执行一次。
- 每个循环都应在边界条件和一般条件下各执行一次。
- 测试程序内部数据结构的有效性等。

4. 软件测试步骤

软件测试实际上可分成 4 步进行。

1) 单元测试

单元测试也称为模块测试,在模块编写完成且无编译错误后就可以进行。如果选用机器测试,一般用白盒测试法,多个模块可以同时进行。

测试一个模块时需要编写一个驱动模块和若干个桩(stub)模块。驱动模块的功能是向被测试模块提供测试数据,驱动被测模块,并从被测模块中接收测试结果。桩模块的功能是模拟被测模块所调用的子模块,它接收被测模块的调用,检验调用参数,模拟被调用的子模块功能,把结果送回被测模块。

2) 组装测试

组装测试也称为集成测试,就是把模块按系统设计说明书的要求组合起来进行测试。即使所有模块都通过了测试,但在组装之后,仍可能会出现問題。例如:穿过模块的数据被丢失;一个模块的功能对其他模块造成有害的影响;各个模块组装起来没有达到预期的功能;全局数据结构出现问题;另外,对于单个模块来说其误差也许可以接受,但模块组合后,可能会出现误差累积,最后到不能接受的程度,所以需要组装测试。

通常,组装测试有两种方法:一种是分别测试各个模块,再把这些模块组合起来进行

整体测试,即非增量式集成;另一种是把下一个要测试的模块组合到已测试好的模块中,测试完后再将下一个需要测试的模块组合起来,进行测试,逐步把所有模块组合在一起,并完成测试,即增量式集成。非增量式集成可以对模块进行并行测试,能充分利用人力,并加快工程进度。但这种方法容易混乱,出现错误不容易查找和定位。增量式测试的范围一步步扩大,错误容易定位,而且已测试的模块可在新的条件下再测试,使测试更彻底。

3) 确认测试

经过组装测试之后,软件就被集成起来,接口方面的问题已经解决,将进入软件测试的最后一个环节——确认测试。确认测试的任务就是进一步检查软件的功能和性能是否与用户要求的一致。系统方案说明书描述了用户对软件的要求,所以是软件有效性验证的标准,也是确认测试的基础。

确认测试,首先要进行有效性测试以及软件配置审查,然后进行验收测试和安装测试,经过管理部门的认可和专家鉴定后,软件即可以交给用户使用。

4) 系统测试

系统测试是将已经确认的软件、计算机硬件、外设和网络等其他因素结合在一起,进行信息系统的各种组装测试和确认测试,其目的是通过与系统的需求进行比较,发现所开发的系统与用户需求不符或矛盾的地方。

5. 调试

调试的任务就是根据测试时发现的错误,找出原因和具体的位置,进行改正。调试工作主要由程序开发人员进行,谁开发的程序就由谁来进行调试。

目前常用的调试方法有如下几种:

- 试探法。调试人员分析错误的症状,猜测问题的所在位置,利用在程序中设置输出语句,分析寄存器和存储器的内容等手段来获得错误的线索,一步步地试探和分析出错误所在。这种方法效率很低,适合于结构比较简单的程序。
- 回溯法。调试人员从发现错误症状的位置开始,人工沿着程序的控制流程往回跟踪代码,直到找出错误根源为止。这种方法适合于小型程序,对于大规模程序,由于其需要回溯的路径太多而变得不可操作。
- 对分查找法。这种方法主要用来缩小错误的范围,如果已经知道程序中的变量在若干位置的正确取值,可以在这些位置上给这些变量以正确值,观察程序运行的输出结果。如果没有发现问题,则说明从赋予变量一个正确值开始到输出结果之间的程序没有错,问题可能出在其他程序中;否则错误就在所考察的这部分程序中。对含有错误的程序段再使用这种方法,直到把故障范围缩小到比较容易诊断为止。
- 归纳法。归纳法就是从测试所暴露的问题出发,收集所有正确或不正确的数据,

分析它们之间的关系,提出假设的错误原因,用这些数据来证明或反驳,从而查出错误所在。

- 演绎法。根据测试结果,列出所有可能的错误原因。分析已有的数据,排除不可能和彼此矛盾的原因。对余下的原因,选择可能性最大的,利用已有的数据完善该假设,使假设更具体。用假设来解释所有的原始测试结果,如果能解释这一切,则假设得以证实,也就找出了错误;否则,要么是假设不完备或不成立,要么有多个错误同时存在,需要重新分析,提出新的假设,直到发现错误为止。

10.4.4 系统文档

信息系统的文档是系统建设过程的“痕迹”,是系统维护人员的指南,是开发人员与用户交流的工具。规范的文档意味着系统是按照工程化规范开发的,意味着信息系统的质量有了形式上的保障。文档的欠缺、文档的随意性和文档的不规范,极有可能在原来的开发人员流动以后,导致系统不可维护,不可升级,变成了一个没有扩展性、没有生命力的系统。

信息系统的文档不但包括应用软件开发过程中产生的文档,还包括硬件采购和网络设计中形成的文档;不但包括上述有一定格式要求的规范文档,也包括系统建设过程中的各种来往文件、会议纪要、会计单据等资料形成的不规范文档。后者是各方谈判甚至索赔的重要依据,不但包括系统实施记录,也包括程序资料 and 培训教程等。

文档在系统开发人员、项目管理人员、系统维护人员、系统评价人员以及用户之间的多种作用如下所述。

- 用户与系统分析人员在系统规划和系统分析阶段通过文档进行沟通。这里的文档主要包括可行性研究报告、总体规划报告、系统开发合同、系统方案说明书等。有了文档,用户就能依次对系统分析员是否正确理解了系统的需求进行评价,如不正确,可以在已有文档的基础上进行修正。
- 系统开发人员与项目管理人员通过文档在项目期内进行沟通。这里的文档主要有系统开发计划(包括工作任务分解表、网络图、甘特图、预算分配表等)、系统开发月报以及系统开发总结报告等项目管理文件。有了这些文档,不同阶段之间的开发人员就可以进行工作的顺利衔接,同时还能降低因为人员流动带来的风险,因为接替人员可以根据文档理解前面人员的设计思路或开发思路。
- 系统测试人员与系统开发人员通过文档进行沟通。系统测试人员可以根据系统方案说明书、系统开发合同、系统设计说明书、测试计划等文档对系统开发人员所开发的系统进行测试。系统测试人员再将评估结果撰写成系统测试报告。
- 系统开发人员与用户在系统运行期间进行沟通。用户通过系统开发人员撰写的

文档运行系统。这里的文档主要是用户手册和操作指南。

- 系统开发人员与系统维护人员通过文档进行沟通。这里的文档主要有系统设计说明书和系统开发总结报告。有的开发总结报告写得很详细,分为研制报告、技术报告和技术手册3个文档,其中的技术手册记录了系统开发过程中的各种主要技术细节。这样,即使系统维护人员不是原来的开发人员,也可以在这些文档的基础上进行系统的维护与升级。
- 用户与维修人员在运行维护期间进行沟通。用户在使用信息系统的过程中,将运行过程中的问题进行记载,形成系统运行报告和维修修改建议。系统维护人员根据维护修改建议以及系统开发人员留下的技术手册等文档,对系统进行维护和升级。

10.4.5 系统转换

在进行新老系统转换以前,首先要进行新系统的试运行。在系统测试、调试中,使用的是系统测试数据,有些实际运行中可能出现的问题,很难通过这些数据被发现。所以,一个系统开发后,让它实际运行一段时间,是对系统最好的检验和测试方法。

系统试运行阶段的主要工作有:

- 对系统进行初始化、输入各原始数据记录。
- 记录系统运行的数据和状况。
- 核对新系统和老系统(人工或计算机系统)输出的结果。
- 对实际系统的输入方式进行考察(是否方便,效率如何,安全可靠,误操作保护等)。
- 对系统实际运行的响应速度(包括运算速度、传递速度、查询速度、输出速度等)进行实际测试。

新系统试运行成功之后,就可以在新系统和老系统之间互相转换。新旧系统之间的转换方式有直接转换、并行转换和分段转换。

- 直接转换。直接转换就是在确定新系统运行无误时,立刻启用新系统,终止老系统运行。这种方式可以节省人员和设备费用,适用于一些处理过程不太复杂,数据不很重要的场合。
- 并行转换。这种转换方式指的是新老系统并行工作一段时间,经过一段时间的考验以后,新系统正式替代老系统。对于较复杂的大型系统,它提供了一个与旧系统运行结果进行比较的机会,可以对新旧两个系统的时间要求、出错次数和工作效率给以公正的评价。由于与旧系统并行工作,消除了尚未认识新系统之前的紧张和不安。在银行、财务和一些企业的核心系统中,这是一种经常使用的切换方

式。它的主要特点是安全可靠,但费用和工作量都很大。

- 分段转换。分段转换又称逐步转换、向导转换或试点过渡法等。这种切换方式实际上是以上两种切换方式的结合。在新系统全部正式运行前,一部分一部分地代替老系统。那些在转换过程中还没有正式运行的部分,可以在一个模拟环境中继续试运行。这种方式既保证了可靠性,又不至于费用太大。但是这种分段转换要求子系统之间有一定的独立性,对系统的设计和实现都有一定的要求,否则就无法实现这种分段转换的设想。

在实际工作中,切换方法较为灵活。一个信息系统从使用到成熟再到提高,是一个比较长的过程。只有遵循数据处理的阶段性,信息系统才能健康发展。现以一个连锁企业开始实施新系统为例,介绍信息系统的成长过程。

(1) 初始阶段:企业首先为应用系统做基本资料的准备,进行总部和“配送”核心系统的实施。

(2) 推广阶段:总部和“配送”系统稳定后,先从1~2家门市试点开始,以门市核心模块为主,完成门市与总部的信息交换及物流过程;然后再逐步推广门市系统,直至完成所有门市的联网工作。

(3) 控制阶段:所有门市联网完成后,进行准确、及时的基本数据采集和调整工作。

(4) 集成阶段:再考虑自动补货、自动配货、财务接口等高级模块的应用。

(5) 管理阶段:进入数据的全面启用和管理决策阶段。最后,系统步入成熟阶段。

实际上,每个企业在不同阶段的发展过程中,对具体问题的解决方法是不同的,随时都会进行以上阶段的周期重复,随着每次重复时起点的不断升高,整个企业的数据应用水平也就随之逐步提高了。

10.5 系统运行和维护知识

10.5.1 系统维护

系统的可维护性可以定性的定义为维护人员理解、改正、改动和改进这个软件的难易程度。提高可维护性是开发管理信息系统所有步骤的关键目的,系统是否能被很好地维护,可用系统的可维护性这一指标来衡量。

1. 系统的可维护性的评价指标

- 可理解性。指别人能理解系统的结构、界面功能和内部过程的难易程度。详细的设计文档、模块化和结构化设计以及良好的高级程序设计语言等,都有助于提高可理解性。

- 可测试性。诊断和测试的容易程度取决于易理解的程度。好的文档资料有利于诊断和测试,同时,程序的结构、高性能的测试工具以及周密计划的测试工序也是至关重要的。为此,开发人员在系统设计和编程阶段就应尽力把程序设计成易诊断和测试的。此外,在系统维护时,应该充分利用系统调试阶段保存下来的调试用例。
- 可修改性。诊断和测试的容易程度与系统设计所制定的设计原则有直接关系。模块的耦合、内聚、作用范围与控制范围的关系等,都对可修改性有影响。

2. 文档与软件维护

文档是软件可维护性的决定因素。由于长期使用的大型软件系统在使用过程中必然会经受多次修改,所以文档显得非常重要。

软件系统的文档可以分为用户文档和系统文档两类。用户文档主要描述系统功能和使用方法,并不关心这些功能是怎样实现的;系统文档描述系统设计、实现和测试等各方面的内容。

可维护性是所有软件都应具有的基本特点,必须在开发阶段保证软件具有可维护的特点。在软件工程的每一个阶段都应考虑并提高软件的可维护性,在每个阶段结束前的技术审查和管理复查中,应该着重对可维护性进行复审。

在系统分析阶段的复审过程中,应该对将来要改进的部分和可能会修改的部分加以注解并指明,并且指出软件的可移植性问题以及可能影响软件维护的系统界面;在设计阶段的复审期间,应该从容易修改、模块化和功能独立的目的出发,评价软件的结构和过程;在系统实施阶段的复审期间,代码复审应该强调编码风格和内部说明文档这两个影响可维护性的因素。在完成了每项维护工作之后,都应该对软件维护本身进行认真的复审。

3. 软件文档的修改

维护应该针对整个软件配置,不应该只修改源程序代码。如果对源程序代码的修改没有反映在设计文档或用户手册中,可能会产生严重的后果。每当对数据、软件结构、模块过程或任何其他有关的软件特点做改动时,必须立即修改相应的技术文档。不能准确反映软件当前状态的设计文档可能比完全没有文档更坏。在以后的维护工作中很可能因文档不完全符合实际而不能正确理解软件,从而在维护中引入过多的错误。

4. 系统维护的内容及类型

系统维护主要包括硬件设备的维护、应用软件的维护和数据的维护。

1) 硬件维护

硬件的维护应由专职的硬件维护人员负责。主要有两种类型的维护活动,一种是定期的设备保养性维护,保养周期可以是一周或一个月不等,维护的主要内容是进行例行的

设备检查与保养,易耗品的更换与安装等;另一种是突发性的故障维护,即当设备出现突发性故障时,由专职的维修人员或请厂方的技术人员来排除故障,这种维修活动所花时间不能过长,以免影响系统的正常运行。

2) 软件维护

软件维护主要是指根据需求变化或硬件环境的变化对应用程序进行部分或全部的修改。修改时应充分利用源程序,修改后要填写程序修改登记表,并在程序变更通知书上写明新老程序的不同之处。

软件维护的内容一般包括正确性维护、适应性维护、完善性维护和预防性维护。

3) 数据维护

数据维护工作主要由数据库管理员负责,主要负责数据库的安全性和完整性以及并发性控制。数据库管理员还要负责维护数据库中的数据。当数据库中的数据类型及长度等发生变化时,或者需要添加某个数据项时,要负责修改相关的数据库和数据字典,并通知有关人员。另外数据库管理员还要负责定期更新数据字典文件及一些其他数据管理文件,以保留系统运行和修改的轨迹。当系统出现硬件故障并得到排除后要负责数据库的恢复工作。

数据维护中还有一项很重要的内容,那就是代码维护。不过代码维护发生的频率相对较小。代码的维护应由代码管理小组进行。变更代码应经过详细讨论,确定之后要用书面形式推行。代码维护的困难往往不在于代码本身的变更,而在于新代码的推行。为此,除了成立专门的代码管理小组外,各业务部门要指定专人进行代码管理,通过他们推行使用新代码。这样做的目的是要明确管理职责,有助于防止和更正错误。

10.5.2 系统评价

1. 系统评价的目的和任务

信息系统的评价分为广义和狭义两种。广义的信息系统评价是指从系统开发的开始到结束的每一阶段都需要进行评价。狭义的信息系统评价则是指在系统建成并投入运行之后所进行的全面和综合的评价。

按评价的时间与信息系统所处的阶段的关系,又可从总体上把广义的信息系统评价分成立项评价、中期评价和结项评价。

1) 立项评价

立项评价指信息系统方案在系统开发前的预评价,即系统规划阶段中的可行性研究。评价的目的是决定是否立项进行开发,评价的内容是分析当前开发新系统的条件是否具备,明确新系统目标实现的重要性和可能性,主要包括技术上的可行性、经济上的可行性、管理上的可行性和开发环境的可行性等方面。由于事前评价所用的参数大都是不确定

的,所以评价的结论具有一定的风险性。

2) 中期评价

项目中期评价包含两种含义,一是指项目方案在实施过程中,因外部环境出现重大变化,比如市场需求变化,竞争性技术或更完美的替代系统的出现,或者发现原先设计有重大失误等,需要对项目的方案进行重新评估,以决定是继续执行还是终止该方案;另一种含义也可称为阶段评估,是指在信息系统开发正常情况下,对系统设计、系统分析和系统实施阶段的阶段性成果进行评估。由于一般都将阶段性成果的提交视为信息系统建设的里程碑,所以,阶段评估又可称为里程碑式评价。

3) 结项评价

信息系统的建设是一个项目,是项目就需要有终结时间。结项评价是指项目准备结束时对系统的评价,一般是在信息系统正式运行以后,为了解系统是否达到预期的目的和要求而对系统运行的实际效果进行的综合评价。所以,结项评价又是狭义的信息系统评价。信息系统项目的鉴定是结项评价的一种正规的形式。结项评价的主要内容包括系统性能评价、系统的经济效益评价以及企业管理效率提高和管理水平改善以及管理人员劳动强度减轻等间接效果评价。通过结项评价,用户可以了解系统的质量和效果,检查系统是否符合预期的目的和要求;开发人员可以总结开发工作的经验、教训,对今后的工作十分有益。

2. 系统评价指标

系统评价指标包括系统质量、技术水平、运行质量、用户需求、系统成本、系统效益和财务评价。

10.5.3 系统运行管理

1. 运行管理制度

一个规范管理的企业,每一项具体的业务都有一套科学的运行制度。信息系统也不例外,同样需要一套管理制度,以确保信息系统的正常和安全运行。

1) 各类机房安全运行管理制度

信息系统的运行管理制度首先要确保机房必须处于监控之中。机房安全运行管理制度应该包括如下主要内容:

- 身份登记与出入验证。
- 带入带出物品检查。
- 参观中心机房必须经过审查。
- 专人负责启动、关闭计算机系统。
- 对系统运行状况进行监视,跟踪并详细记录运行信息。

- 对系统进行定期保养和维护。
- 操作人员在指定的计算机或终端上操作,对操作内容按规定进行登记。
- 不做与工作无关的操作,不运行来历不明的软件。
- 不越权运行程序,不查阅无关参数。
- 发现操作异常立即报告。

2) 信息系统的其他管理制度

此外,还要确保软件、数据、信息等其他要素必须处于监控之中。信息系统的其他管理制度主要包括如下内容:

- 必须有重要的系统软件、应用软件管理制度,如系统软件的更新维护,应用软件的源程序与目标程序分离等。
- 必须有数据管理制度,例如重要输入输出数据的管理。
- 必须有密码口令管理制度,做到口令专管专用,定期更改并在失密后立即报告。
- 必须有网络通信安全管理制度,实行网络电子公告系统的用户登记和对外信息交流的管理制度。
- 必须有病毒的防治管理制度。及时检测和清除计算机病毒,并备有检测和清除的记录。
- 必须有人员调离的安全管理制度。例如,人员调离的同时马上收回钥匙、移交工作、更换口令、取消账号,并向被调离的工作人员申明其保密义务。
- 建立安全培训制度,进行计算机安全法律教育、职业道德教育和计算机安全技术教育。对关键岗位的人员进行定期考核。
- 建立合作制度。加强与相关单位的合作,及时获得必要的信息和技术支持。

除此之外,任何信息系统的运行都必须遵守国家的有关法律和规定,特别是关于计算机信息系统安全的法律规定。

2. 日常运行管理内容

信息系统的日常运行管理是为了保证系统能长期有效地正常运转,包括系统运行情况的记录、系统运行的日常维护等工作。

对系统运行情况的记录应事先制定登记格式和登记要点,人工记录的系统运行情况和系统自动记录的运行信息,都应作为基本的系统文档按照规定的期限保管。这些文档既可以在系统出现问题时查清原因和责任,还能作为系统维护的依据和参考。

1) 系统运行情况的记录

原则上讲,从每天计算机的打开、应用系统的进入、功能项的选择与执行,到下班前的数据备份、存档、关机等,都要对系统软硬件及数据等的运作情况作记录。运行情况有正常、不正常与无法运行 3 种情况。正常情况可不予记录,对于不正常和无法运行的情况则

应将出现的现象、发生的时间及可能的原因作尽量详细的记录。因为这些信息对系统问题的分析与解决有重要的参考价值。

2) 审计踪迹

审计踪迹(audit trail)是指系统中设置了自动记录功能,能通过自动记录的信息发现或判明系统的问题和原因。这里的审计有两个特点,一是每日都进行,二是主要进行技术方面的审查。

在审计踪迹系统中,建立审计日志是一种基本的方法。通过日志,系统管理员可以了解到有哪些用户在什么时间、以什么样的身份登录到系统,也可以查到对特定文件和数据所进行的改动。

现在大多数的操作系统和数据库都提供跟踪并自动记录的功能,一些数据库系统中还提供审计踪迹数据字典,可以用预先定义的审计踪迹数据字典视图来观察审计踪迹数据。

3) 审查应急措施的落实

为了减少意外事件引起的对信息系统的损害,首先要制定应付突发性事件的应急计划,然后每日要审查应急措施的落实情况。

应急计划主要针对一些突发性的、灾害性的事件,例如火灾、水害等。因此,机房值班员每日都应仔细审查相应器材和设备是否良好,相应的资源是否做好了备份。

资源备份包括两个方面的工作,即数据备份和设备备份。数据备份是必须要做的,在关键的领域,还必须进行设备备份。应将备份文件拷贝到远离主机或文件中心的其他主机或者存储库中,保证备份文件处在灾难事件影响不到的地方。

4) 系统资源的管理

在维护信息系统正常运行过程中还应对计算机的使用及打印机、墨粉的消耗等制定合理的管理办法。

对不能充分满足用户需求的资源,一般可采用收费的方法来控制。收费既要能使系统的效能发挥到最大,使得信息系统部门的利益和管理措施得到保证,又要做到业务部门愿意接受,不能妨碍业务部门的正常使用。

5) 系统软件及文档管理

系统软件及文档管理的内容包括:

(1) 系统软件的管理除日常维护以外,还包括版本更新和升级等。

由于计算机科学技术的迅速发展,新的硬软件不断推出,使系统的外部环境发生变化。这里的外部环境不仅包括计算机硬件软件的配置,还包括数据库或数据存储方式在内的数据环境。为了适应企业的发展,版本更新和升级必不可少。

(2) 对信息系统文档的管理。

信息系统的文档与其他类型的文档一样,也具有它自身的生命周期,分为创建期、处理期、存储期、使用期、销毁期。每种文档都处于生命周期中的某一时期。当然,周期的划分也不是绝对的,各周期有时是不能截然分开的。信息系统文档的生命周期普遍要比信息系统的生命周期长。也就是说,绝大多数信息系统的文档要在相应的信息系统淘汰3~5年后才能销毁。

为了最终得到高质量的信息系统文档,在信息系统的建设过程中必须加强对文档的管理。文档管理应从以下几个方面着手进行。

- 文档管理的制度化。
- 文档要标准化、规范化。
- 文档管理的人员保证。
- 维护文档的一致性。
- 维持文档的可追踪性。

第 11 章 数据库设计

数据库设计涉及的知识面广,研制的周期长,是一门综合性的技术,包括数据库的基本知识和设计技术、计算机科学基础知识及程序设计技巧、软件工程的原理和方法以及应用领域的知识。数据库设计是指对于一个给定的应用环境,构造最优的数据库模式,建立数据库及其应用系统,使之能有效地存储数据,满足各种用户的需求(信息要求和处理要求)。

11.1 数据库设计概述

数据库设计属于系统设计的范畴。通常把使用数据库的系统统称为数据库应用系统,把数据库应用系统的设计简称为数据库设计。

按照软件工程对系统生命周期的定义,软件生命周期分为制定计划、需求分析、设计、程序编制、测试以及运行维护 6 个阶段。在数据库设计中也参照这种划分,把数据库应用系统的生命周期分为数据库规划、需求收集与分析、数据库设计与应用程序设计、实现、测试以及运行维护 6 个阶段。

1) 数据库规划

数据库规划是创建数据库应用系统的起点,是数据库应用系统的任务陈述和任务目标制定阶段。任务陈述定义数据库应用系统的主要目标,而每个任务目标定义系统必须支持的特定任务。数据库规划过程还必然包括工作量估计、使用的资源和需要的经费等。同时还应当定义系统的范围和边界以及它与公司信息系统其他部分的接口。

2) 需求收集与分析

需求收集与分析是以用户的角度,从系统中的数据和业务规则入手,收集和整理用户的信息,以特定的方式加以描述,是下一步工作的基础。

3) 数据库的设计

数据库的设计是对用户数据的组织和存储设计,应用程序设计是在数据库设计的基础上对数据操作及业务实现的设计,包括事务设计和用户界面设计。

4) 数据库系统实现

数据库系统实现是依照设计,使用 DBMS 支持的数据定义语言(DDL)实现数据库的建立,用高级语言(Basic、Delphi、C、C++、Power builder 等)编写应用程序。

5) 测试阶段

测试阶段是在数据系统投入使用之前,通过精心制定的测试计划和测试数据来测试系统的性能是否满足设计要求,从而发现问题。

6) 运行维护

数据库应用系统经过测试和试运行后即可正式投入运行。运行维护指的是系统投入使用后,必须不断地对其进行评价、调整与修改,直至系统消亡。

11.2 系统需求分析

系统需求分析是在项目确定之后,用户和设计人员对数据库应用系统所涉及的内容(数据)和功能(行为)的整理和描述,是以用户的角度来认识系统。这一过程是后续开发的基础,以后的逻辑设计和物理设计以及应用程序的设计都会以此为依据。如果这一阶段的工作没有做好,势必会为以后的工作带来困难,甚至要再重新回过头来做需求分析,影响整个项目的工期,在人力、物力等方面造成浪费。因此,这一阶段的工作要求做到耐心细致,这是整个设计开发过程中最困难、最耗时的一步。

11.2.1 需求分析的任务和目标

需求分析阶段的任务是:对现实世界要处理的对象(组织、部门、企业等)进行详细调查,在了解现行系统的概况,确定新系统功能的过程中,收集支持系统目标的基础数据及处理方法。需求分析是在用户调查的基础上,通过分析,逐步明确用户对系统的需求,包括数据需求和围绕这些数据的业务处理需求,以及对数据安全性和完整性方面的要求。

在需求分析的过程中,首先应当确定系统范围。在绝大多数情况下,用户并非计算机专业人员,对计算机并不很了解,用户总希望所开发的系统能够尽可能多地实现他们要想要的功能,而有些是目前不可能实现的。其次,企业或部门目前可能已经有现存的系统在运行,但不能满足用户的要求。在新的系统中,应该继承现有系统中的数据。现存系统也可能会作为新系统中的一部分继续运行,这些都必须明确。再者,要充分考虑用户的应用需求。随着企业的发展,对一些可预见的需求也应当加以考虑,使新系统能够有一定的灵活性和可扩充性,以适应未来的发展,而不仅仅是满足当前的应用需求。

需求分析阶段是以调查和分析为主要手段的,以此满足用户对系统的下列要求。

(1) 信息要求:用户需要在系统中保存哪些信息,从这些保存的信息中要得到什么样的信息,这些信息以及信息间应当满足的完整性要求。

(2) 处理要求:用户在系统中要实现什么样的操作功能,保存信息的处理过程和方式,各种操作处理的频度、响应时间要求、处理方式,以及处理过程中的安全性要求和完整

性要求等。

11.2.2 需求分析的方法和步骤

参与需求分析的主要人员是分析人员和用户,由于数据库应用系统是面向企业和部门的具体业务,分析人员一般并不了解,而同样用户也不会具有系统分析的能力,这就需要双方进行有效的沟通,使得设计人员对用户的各项业务有充分的了解和熟悉,并进行分析和加工,将用户眼中的业务转换为设计人员所需要的信息组织。

了解用户需求的方法就是调查。可以采取开调查会、跟班作业、查阅文献、书面填表、交流询问等方式,对用户的信息需求进行收集。收集的内容包括:数据、业务处理的过程和依据、处理的时间和频度等。

在收集信息的同时,设计人员要对其进行加工和整理,以数据字典和数据流图的形式描述出来,并以设计人员的角度向用户讲述这些信息,根据用户的反馈加以修改并确定。

数据字典是对用户信息要求的整理和描述。信息需求定义了未来信息系统用到的所有信息,包括用户将向数据库中输入什么信息,从数据库中要得到什么信息,各类信息的内容和结构,信息之间的联系等。数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程5个部分。

对用户处理要求的描述采用数据流图的形式,即对数据采取什么样的加工方式和操作,得到用户需要的结果,通常是对业务处理过程的描述。数据流图是应用程序设计的基础,有关数据流图的详细内容可参阅本书第十章。

下面以一个机械制造厂的采购业务为例来理解需求分析的方法和步骤。

1) 数据字典

(1) 数据项:数据项是数据的最小单位。对数据项的描述一般包括项名、含义说明、别名、类型、长度、取值范围及该项与其他项的逻辑关系,常以表格的形式给出。如采购业务中订货单的订单号,其数据项的描述如下所示。

数据项名: 订货单号

说 明: 用来惟一标识每张定货单

类 型: 字符型

长 度: 8

别 名: 采购单号

取值范围: 00000001~99999999

(2) 数据结构: 数据结构是若干有意义的数据项的集合,用以表示某一具体的事物,

包括数据结构名、含义和组成成分等。如对采购单数据结构的描述。

数据结构：采购单

含 义：记录采购信息，包括采购什么材料及其数据

组成成分：采购单号

材料名称

数量

(3) 数据流：数据流可以是数据项，也可以是数据结构，表示某一次处理的输入输出数据，包括数据流名、说明、数据来源、数据去向及需要的数据项或数据结构，如采购计划数据流。

数据流名：采购计划

说 明：根据生产需要的原材料，选定供应商，编制采购计划

来 源：原材料需求表

去 向：采购单

数据结构：原材料需求表

供应商

(4) 数据存储：加工中需要存储的数据，包括数据存储名、说明、输入数据流、输出数据流、组成成分、数据量、存取方式以及存取频度等。如原材料的价目表，在计算成本和支付采购费用的处理过程中要用到这些数据。

数据存储名：原材料价目表

说 明：记录每一原材料的名称、供应商及价目，在计算产品成本和采购费用支付处理中使用
输入数据流：订购单

输出数据流：支付费用表

数据描述： 原材料名称

供应商

单价

数 据 量：约 50 条记录

存取方式： 随机

存取频度： 30 次/月

(5) 处理过程：加工处理过程的定义和说明，包括处理名称、输入数据、输出数据、数据存储及响应时间等，如采购支付处理。

处理过程名：采购支付

说明：根据采购单、原材料价目表，计算出应付原材料采购费用

输入数据：采购单

数据存储：原材料价目表

输出数据：支付费用表

2) 数据流图

数据流图的具体例子如图 11-1 所示。

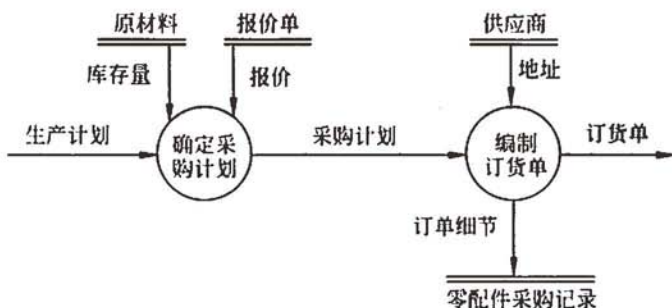


图 11-1 采购流程图

需求分析阶段的成果是系统需求说明书，主要包括数据流图、数据字典、各种说明性表格、统计输出表以及系统功能结构图等。系统需求说明书是以后设计、开发、测试和验收等过程的重要依据。

11.3 概念结构设计

数据库概念结构设计阶段是在需求分析的基础上，依照需求分析中的信息要求，对用户信息加以分类、聚集和概括，建立信息模型，并依照选定的数据库管理系统软件，把它们转换为数据的逻辑结构，再依照软硬件环境，最终实现数据的合理存储。这一过程也称为数据建模。

这一过程可分解为 3 个阶段：概念结构设计、逻辑结构设计和物理结构设计。

11.3.1 概念结构设计策略与方法

概念结构设计是设计人员以用户的观点,对用户信息的抽象和描述。从认识论的角度来讲,是从现实世界到信息世界的第一次抽象,并不考虑具体的数据库管理系统。

现实世界的事物纷繁复杂,即使是某一具体应用,由于存在大量不同的信息和对信息的各种处理,也必须加以分类整理,理清各类信息之间的关系,描述信息处理的流程,这一过程就是概念结构设计。

概念结构设计的策略通常有以下 4 种:

- 自顶向下。即首先定义全局概念结构的框架,然后逐步细化。
- 自底向上。即首先定义各局部应用的概念结构,然后将它们集成起来,得到全局概念结构。
- 逐步扩张。即首先确定核心业务的概念结构,然后以此为中心向外扩张,最终实现全局概念结构。
- 混合策略。即将自顶向下和自底向下两种策略结合使用,首先确定全局框架,划分若干个局部概念模型,再采取自底向上的策略实现各局部概念模型,加以合并,最终实现全局概念模型。

实际应用中,这些策略并没有严格的限定。可以根据具体业务的特点选择,如对于组织机构管理,因其固有的层次结构,可采用自顶向下的策略;对于已实现计算机管理的业务,通常可以以此为核心,采取逐步扩张的策略。

概念结构设计最著名最常用的方法是 P. P. S Chen 于 1976 年提出的实体-联系方法(Entity-Relationship Approach),简称 E-R 方法。它采用 E-R 模型将现实世界的信息结构统一由实体、属性以及实体之间的联系来描述。

使用 E-R 方法时,无论是哪种策略,都要对现实事物加以抽象,以 E-R 图的形式描述出来。

对现实事物抽象的 3 种方法分别是分类、聚集和概括。

- 分类(classification): 对现实世界的事物,按照其固有的共同特征和行为,定义一种类型。这在现实生活中很常见,如学校中的学生和教师就属于不同的类型。在某一类型中,个体是类型的一个成员或实例,即“is member of”。如李娜是学生类型中的一个成员。
- 聚集(aggregation): 定义某一类型所具有的属性。如学生类型具有学号、姓名、性别和班级等共同属性。每一个学生都是这一类型中的个体,通过这些属性上的不同取值来区分。各个属性是所属类型的一个成分,即“is part of”。如姓名是学生类型的一个成分。

- 概括(generalization): 由一种已知类型定义新的类型。如由学生类型定义研究生类型,在学生类型的属性上增加导师等其他属性就构成研究生类型。通常把已知类型称为超类(superclass),新定义的类型称为子类(subclass)。子类是超类的一个子集,即“is subset of”。如研究生是学生的一个子集。

11.3.2 用 E-R 方法建立概念模型

E-R 图的设计要依照上述的抽象机制,对需求分析阶段所得到的数据进行分类、聚集和概括,确定实体、属性和联系。具体步骤如下:

- ① 选择局部应用;
- ② 逐一设计分 E-R 图;
- ③ E-R 图合并。

1. 选择局部应用

需求分析阶段会得到大量的数据,这些数据分散杂乱,许多数据会应用于不同的处理,数据与数据之间的关联关系也较为复杂。要最终确定实体、属性和联系,就必须根据数据流图这一线索,理清数据。

数据流图是对业务处理过程从高层到底层的一级级抽象,高层抽象流图一般反映系统的概貌,对数据的引用较为笼统;而底层又可能过于细致,不能体现数据的关联关系。因此,要选择适当层次的数据流图,让这一层的每一部分对应一个局部应用,实现某一项功能。从这一层入手,就能很好地设计分 E-R 图。

2. 逐一设计分 E-R 图

划分好各个局部应用之后,就要对每一个局部应用逐一设计分 E-R 图,又称为局部 E-R 图。

对于每一局部应用(其所用到的数据这时都应已收集在数据字典中了),依照该局部应用的数据流图,从数据字典中提取出数据,使用抽象机制,确定局部应用中的实体、实体的属性、实体标识符和实体间的联系及其类型。

事实上,在形成数据字典的过程中,数据结构、数据流和数据存储都是根据现实事物来确定的,因此都已经基本上对应了实体及其属性。以此为基础,加以适当调整,增加联系及其类型,就可以设计分 E-R 图。

现实生活中的许多事物,是作为实体还是属性并没有明确的界定,这需要根据具体情况而定,一般应遵循以下两条准则:

- (1) 属性不可再分,即属性不再具有需要描述的性质,不能有属性的属性;
- (2) 属性不能与其他实体发生联系,联系是实体与实体间的联系。

3. E-R 图合并

根据局部应用设计好各局部 E-R 图之后,就可以对各分 E-R 图进行合并。合并的目的是在合并过程中解决分 E-R 图间存在的冲突,消除分 E-R 图之间存在的信息冗余,使之成为能够被整个系统所有用户都理解和接受的统一的、精炼的全局概念模型。合并的方法是将具有相同实体的两个或多个 E-R 图合而为一,在合成后的 E-R 图中把相同实体用一个实体表示。合成后的实体的属性是所有分 E-R 图中该实体的属性的并集,并以此实体为中心,并入其他所有分 E-R 图。再把合成后的 E-R 图以分 E-R 图看待,合并剩余的分 E-R 图,直至把所有的 E-R 图全部合并,最终构成一张全局 E-R 图。

分 E-R 图之间的冲突主要有以下 3 类。

(1) 属性冲突:同一属性可能会存在于不同的分 E-R 图,由于设计人员不同或是出发点不同,属性的类型、取值范围及数据单位等可能会不一致,这些属性对应的数据将来只能以一种形式在计算机中存储,这就需要在设计阶段进行统一。

(2) 命名冲突:相同意义的属性,在不同的分 E-R 图上有不同的命名,或者是名称相同的属性在不同的分 E-R 图中代表着不同的意义,这些也要进行统一。

(3) 结构冲突:同一实体在不同的分 E-R 图中有不同的属性,同一对象在某一分 E-R 图中被抽象为实体而在另一分 E-R 图中又被抽象为属性,这些也需要统一。

在分 E-R 图的合并过程中要对其进行优化,具体可以从以下几个方面进行。

(1) 实体类型的合并:两个具有 1:1 联系或 1:n 联系的实体,可以予以合并,使实体个数减少,有利于减少将来数据库操作过程中的连接开销。

(2) 冗余属性的消除:各分 E-R 图中的属性一般不存在冗余,但合并后就可能出现冗余。因为合并后的 E-R 图中的实体继承了合并前该实体在分 E-R 图中的全部属性,而属性间可能存在冗余,即某一属性可以由其他属性确定。

(3) 冗余联系的消除:在分 E-R 图合并过程中,可能会出现实体联系的环状结构,即某一实体 A 与另一实体 B 间有直接联系,同时 A 又通过其他实体与实体 B 发生间接联系。如果直接联系可以通过间接联系表达,可消除直接联系。

对所有的分 E-R 图合并完之后,就形成了整个系统的全局 E-R 图,从而也就完成了概念结构设计。

11.4 逻辑结构设计

概念结构设计与数据模型无关,而一个数据库系统的实现是以具体的 DBMS 为基础的。在概念结构设计完成之后,就要依照选用的 DBMS,对该 DBMS 支持的数据模型相对应的逻辑结构进行设计。逻辑结构设计是在概念结构设计的基础上进行的数据模型设

计,可以是层次、网状模型和关系模型。由于当前的绝大多数 DBMS 都是基于关系模型的,E-R 方法又是概念结构设计的主要方法,如何在全局 E-R 图基础上进行关系模型的逻辑结构设计成为这一阶段的主要内容。在进行逻辑结构设计时不需要考虑数据在某一 DBMS 下的具体物理实现,即数据在计算机中是如何存储的。

逻辑结构设计阶段的主要任务是:

- 确定数据模型;
- 将 E-R 图转换为指定的数据模型;
- 确定完整性约束;
- 确定用户视图。

11.4.1 E-R 图向关系模式的转换

采用 E-R 方法所得到的全局概念模型是对信息世界的描述,并不适用于计算机处理,为适合关系数据库系统的处理,必须将 E-R 图转换成关系模式。E-R 图是由实体、属性和联系三要素构成,而关系模型中只有惟一的结构——关系模式。通常应采用以下方法加以转换。

1. 实体向关系模式的转换

将 E-R 图中的实体逐一转换为一个关系模式,实体名对应关系模式的名称,实体的属性转换成关系模式的属性,实体标识符就是关系的码。

2. 联系向关系模式的转换

E-R 图中的联系有 3 种:一对一联系(1:1)、一对多联系(1:n)和多对多联系($m:n$),针对这 3 种不同的联系,有不同的转换方法:

一对一联系的转换:一对一联系向关系模式进行转换有两种方式。一种方式是将联系转换成一个独立的关系模式,关系模式的名称取联系的名称,关系模式的属性包括该联系所关联的两个实体的码及联系的属性,关系的码取自任一方实体的码;另一种方式是将联系归并到关联的两个实体的任一方,在待归并的一方实体属性集中增加另一方实体的码和该联系的属性即可,归并后的实体码保持不变。

一对多联系的转换:一对多联系向关系模式进行转换有两种方式。一种方式是将联系转换成一个独立的关系模式,关系模式的名称取联系的名称,关系模式的属性取该联系所关联的两个实体的码及联系的属性,关系的码是多方实体的码;另一种方式是将联系归并到关联的两个实体的多方,在待归并的多方实体属性集中增加一方实体的码和该联系的属性即可,归并后的多方实体码保持不变。

多对多联系的转换:多对多联系只能转换成一个独立的关系模式。关系模式的名称取联系的名称,关系模式的属性取该联系所关联的两个多方实体的码及联系的属性,关系

的码是多方实体的码构成的属性组。

通过以上方法,就可以将全局 E-R 图中的实体、属性和联系全部转换为关系模式,建立初始的关系模式。

11.4.2 关系模式的规范化

由 E-R 图转换得来的初始关系模式并不完全符合要求,还会有数据冗余或更新异常存在,这就需要经过进一步的规范化处理。具体步骤如下:

① 根据语义确定各关系模式的数据依赖。在设计的前一阶段,只是从关系及其属性来描述关系模式,并没有考虑到关系模式中的数据依赖。关系模式包含着语义,要根据关系模式所描述的自然语义,写出关系数据依赖。

② 根据数据依赖确定关系模式的范式。由关系的码及数据依赖,根据规范化理论,就可以确定关系模式所属的范式。判定关系模式是否符合要求,即是否达到了 3NF 或 4NF。

③ 如果关系模式不符合要求,要根据关系模式的分解算法对其进行分解,达到 3NF、BCNF 或 4NF。

④ 关系模式的评价及修正。根据规范化理论,对关系模式分解之后,就可以在理论上消除冗余和更新异常。但根据处理要求,可能还需要增加部分冗余以满足处理要求,需要作部分关系模式的处理,包括分解、合并或增加冗余属性,提高存储效率和处理效率。

11.4.3 确定完整性约束

根据规范化理论确定了关系模式之后,还要对关系模式加以约束,包括数据项的约束、表级约束及表间约束。可以参照 SQL 标准来确定不同的约束,如检查约束、主码约束以及参照完整性约束,以保证数据的正确性。

11.4.4 用户视图的确定

确定了整个系统的关系模式之后,还要根据数据流图及用户信息建立视图模式,提高数据的安全性和独立性。

(1) 根据数据流图确定处理过程使用的视图。数据流图是某项业务的处理,使用了部分数据,这些数据可能要跨越不同的关系模式。建立该业务的视图,可以降低应用程序的复杂性,并提高数据的独立性。

(2) 根据用户类别确定不同用户使用的视图。不同的用户可以处理的数据可能只是整个系统的部分数据,而确定关系模式时并没有考虑这一因素。如学校的学生管理,不同

的院系只能访问和处理自己的学生信息,这就需要建立针对不同院系的视图,以达到这一要求。这样做可以在一定程度上提高数据的安全性。

11.5 数据库的物理设计

数据库系统的实现离不开具体的计算机,在实现数据库逻辑结构设计之后,就要确定数据库在计算机中的具体存储。数据库在物理设备上的存储结构与存取方法称为数据库的物理结构,它依赖于给定的计算机系统。为一个给定的逻辑数据模型设计一个最适合应用要求的物理结构的过程,就是数据库的物理设计。

在数据库的物理结构中,数据的基本单位是记录。记录是以文件形式存储的,一条存储记录对应着关系模式中的一条逻辑记录。在文件中还要存储记录的结构,如各字段长度和记录长度等,增加必要的指针及存储特征的描述。

数据库的物理设计也离不开具体的 DBMS,不同的 DBMS 对物理文件存取方式的支持是不同的,并且都会作优化处理。这就需要参照具体 DBMS,根据系统的处理要求和数据的特点来确定物理结构。

一般来说,物理设计应做以下工作:

- 确定数据分布;
- 确定存储结构;
- 确定存取方式。

11.5.1 根据计算机系统的运行环境进行数据分布

随着网络技术的不断发展和普及,企业内部网及互联网的应用越来越广泛,分布式数据管理无论从成本和实用性出发,都是一个很好的选择。从企业计算机应用环境出发,确定数据是集中管理还是分布式管理。如果采用分布式管理,数据如何分布,有以下几个方面的考虑:

(1) 根据不同应用分布数据。企业的不同部门一般会使用不同的数据,将与部门应用相关的数据存储在相应的场地,在不同的场地上处理不同的业务。对于应用涉及多个场地的业务,可以通过网络进行数据处理。

(2) 根据处理要求确定数据的分布。不同的处理要求,会有不同的使用频度和响应时间要求。对于使用频度高、响应时间短的数据,应存储在高速设备上。

(3) 数据的分布存储必然会导致数据逻辑结构的变化。要对关系模式做新的调整,回到数据库逻辑设计阶段做必要的修改。

11.5.2 确定数据的存储结构

存储结构具体指数据文件中记录之间的物理结构。在文件中,数据是以记录为单位存储的,可以是顺序存储、哈希存储、堆存储和 B⁺ 树存储等。要根据数据的处理要求和变更频率,选定合理的物理结构。

为提高数据的访问速度,通常会采用索引技术。在物理设计阶段,要根据数据处理和修改要求,确定数据库文件的索引字段和索引类型。

11.5.3 确定数据的访问方式

数据的访问方式是由其存储结构决定的。采用什么样的存储结构,就使用什么样的访问方式。

11.6 应用程序设计

应用程序设计与开发是数据库应用系统开发的重要组成部分,它应遵循应用软件开发的一般规律,即遵循常规的软件工程的方法。数据库应用系统开发是基于 DBMS 的二次开发,一方面是对用户信息的存储,另一方面就是对用户处理要求的实现,在设计过程中通常把数据存储的设计称为结构设计,处理的实现称为行为设计。在现阶段,还没有一种将两者合一的设计方法。

应用程序设计有两种方法:结构化设计方法和面向对象设计方法。在设计阶段,要从分析入手,得到结构化模型或面向对象模型。

数据库应用程序的设计可以借鉴传统的结构化程序设计方法,使用“输入-处理-输出”模型编写系统结构。这些模型大部分依靠数据库和文件,并且不需要复杂的实时处理。同时也有着广泛的结构化程序设计语言作支持,如 C、Basic、Pascal、FORTRAN 等。

面向对象技术是一门较新的技术,在 20 世纪 80 年代后开始得到广泛的应用,适合于具有实时、交互和事件驱动等特性的情形(如多任务的操作系统)。今天,许多商业软件也是交互和事件驱动的应用程序。所以面向对象技术很快就成为应用软件,甚至是商业软件的开发标准。

11.7 数据库系统的实现

数据库应用系统的实现是根据设计,由开发人员编写代码程序来完成的。包括数据库的操作程序和应用程序。

作为关系数据库标准语言,SQL 已经被大量的 DBMS 系统所使用。SQL 提供了数据定义、数据操纵、数据控制等功能,能够实现对数据库的操作。不同的 RDBMS 都不同程度地实现了对标准 SQL 的支持,但在语法格式上可能有些差异,需要参考具体 RDBMS 的参考手册。

使用 SQL 语言编写的数据库操作程序有如下几类:

- 数据库建立程序,使用 SQL 中的 DDL 语言。
- 数据库操纵程序,使用 SQL 中的 DML 语言。
- 事务处理程序,以事务的形式执行复杂的数据操作。
- 存储过程和触发器程序。

应用程序的编写一般采用高级语言如 C、Basic、Pascal、FORTRAN 等来实现,近年来相继出现了专门针对数据库开发的具有高级语言部分功能的开发环境,如 Power builder 和 Delphi。

高级语言通常用来编写前端应用程序,如输入输出界面,和一些复杂的数据处理。通过采用嵌入式 SQL 或数据库访问接口(API)实现对数据库的操作。嵌入式 SQL 由于其编程的复杂性,近年来已逐渐被 ODBC 和 ADO 等接口技术所取代。

11.8 系统实施与维护

在完成数据库的设计和应用程序的设计之后,开发人员应该根据设计的内容,用选定的 RDBMS 提供的 SQL 语言及其他高级语言按设计编写代码,经过调试产生目标模式,然后组织数据入库。

系统实施阶段的主要工作有以下几点。

1. 建立实际的数据库结构

将用 RDBMS 提供的数据库定义语言(DDL)编写的代码,经调试运行后,就可建立系统数据库的结构,包括数据库及基本表、索引和约束等数据库对象。

2. 装载测试数据试运行

数据库结构建立起来之后,应该装载实验数据,进入数据库系统的试运行阶段。这一阶段应通过运行应用程序,执行对数据库的各种操作,测试应用程序的各项功能。测量系统的各项性能指标,分析是否实现预定的设计目标。测试工作一般由数据库设计人员、应用开发人员和用户联合进行,又称为联合测试。

测试阶段应记录出现的各种问题,采用回溯的方式改进和完善设计及程序,以解决出现的问题。

3. 装载数据

经过系统的运行测试,在各项功能都已经实现,系统性能也达到要求的情况下,就可以卸载实验数据,加载用户数据,使系统正式运行。

用户数据可能是以前旧系统的数据,并不完全满足新系统的数据要求,需要进行处理,同时还要做好新系统的数据库的转储和恢复工作,以免发生故障时丢失数据。

11.9 数据库的保护

运行中的数据库系统很容易受到来自多方面的干扰和破坏。如软、硬件系统故障,合法用户的误操作,非法入侵等。

数据库的保护就是要排除和防止各种对数据库的干扰破坏,确保数据安全可靠,以及在数据库遭到破坏后尽快地恢复正常。数据库的保护是通过对数据库的恢复、安全性控制、完整性控制和并发控制 4 个方面来实现的。

11.9.1 事务的概念

事务是一系列的数据库操作,是数据库应用程序的基本逻辑单位。应用程序对数据库的操作都应该以事务的方式进行。

事务(transaction)是用户定义的一个数据库操作序列,这些操作要么全做要么全不做,是一个不可分割的工作单位。事务通常由数据库操纵语言或其他高级语言(如 SQL、CoBOL、C、C++ 和 Java 等)书写的用户程序实现。在关系数据库中,一个事务可以是一条或多条 SQL 语句,也可以是高级语言中的一段程序代码甚至整个程序。在 SQL 标准中定义了有关事务操作的语句,如以 BEGIN TRANSACTION 开始事务,以 END TRANSACTION 结束事务,以 ROLLBACK 进行事务回滚,以 COMMIT 提交事务。

典型的例子是银行转账业务,从账户 A 转入账户 B 金额 x 元,从顾客角度来看,转账是一次单独操作。而在数据库系统中,它至少是由两个操作组成的:从账户 A 减去 x 元,给账户 B 加上 x 元。

事务具有以下 4 个特性:

- 原子性(atomicity):在数据库中事务的所有操作要么全做要么全都不做。如银行转账中的两个操作必须作为一个单位来处理,不能只执行部分操作。
- 一致性(consistency):一个事务独立执行的结果,将保持数据的一致性,即数据不会因为事务的执行而遭受破坏。数据的一致性是对现实世界的真实状态的描述,如银行转账业务执行后账目应该是平衡的。数据库在运行过程中会出现瞬间的不一致状态,如从 A 账户减去 x 元到给 B 账户加上 x 元之前这段时间数据是不

一致,但这种不一致只能出现在事务执行过程中,并且不一致的数据不能被其他事务所访问。一致性可以由 DBMS 的完整性约束机制来自动完成,而复杂的事务则由应用程序来完成。

- 隔离性(isolation): 一个事务的执行不能被其他事务干扰。并发事务在执行过程中可能会对同一数据进行操作,这些事务的操作不应该相互干扰,是相互隔离的。如事务执行过程中数据不一致性状态出现时不能让其他事务读取到不一致的数据。
- 持久性(durability): 一个事务一旦提交,它对数据库的改变必须是永久的,即便系统出现故障时也是如此。如转账事务执行成功后,A、B 两个账户上的余额就是一个新的值,在没有出现下一个事务对其修改之前一直保持不变,即使系统出现故障,也应该恢复到这个值。

这四个特性通常被称为事务的 ACID 特性,这一缩写取自四个特性的英文首字母。

下面是银行转账事务的伪代码:

```
BEGIN TRANSACTION
    read(A);                /* 读账户 A 的余额 */
    A=A-x;
    IF(A<0) THEN
        print("金额不足,不能转账");
        ROLLBACK;          /* 撤销该事务,回到事务执行前的状态 */
    ELSE
        write(A);           /* 写入账户 A 的余额 */
        read(B);
        B=B+1;
        write(B);
        COMMIT;             /* 提交事务 */
    ENDIF;
END TRANSACTION
```

11.9.2 数据库的备份与恢复

在数据库的运行过程中,难免会出现计算机系统的软、硬件故障,这些故障会影响数据库中数据的正确性,甚至破坏数据库,使数据库中的全部或部分数据丢失。因此,数据库的关键技术在于建立冗余数据,即备份数据。在系统出现故障后能够及时使数据库恢复到故障前的正确状态,就是数据库恢复技术。

1. 数据库备份的必要性

事实上,在当今的信息社会,最珍贵的财产并不是计算机软件,更不是计算机硬件,而

是企业长期发展过程中所积累下来的业务数据。建立网络最根本的用途就是要更加方便地传递与使用数据。但人为错误、硬盘损坏、电脑病毒、断电或是天灾人祸等都有可能造成数据的丢失。所以应该强调指出：“数据是资产，备份最重要”。备份意识实际上就是数据的保护意识，在危机四伏的网络环境中，数据随时有被毁坏的可能。系统灾难的发生，不是是否会，而是迟早的问题。造成系统数据破坏、丢失的原因很多，有些还往往被人们忽视。正确分析威胁数据安全的因素，及时地备份数据，能使系统的安全防护更有针对性。数据库系统中可能发生的故障有很多种，大致可以将故障分为以下几类。

1) 事务故障

事务故障是由于事务程序内部错误而引起的。这些错误有些是可以预期的，如银行转账中的金额不足，有些则是不可预期的，如非法输入、运算溢出等。对于可预期的错误，应该由应用程序以回滚的方式来恢复，非预期的故障应用程序无法处理，而是由 DBMS 系统来实现故障恢复的。如非法输入由约束机制检查并恢复。事务故障通常指非预期的故障。

2) 系统故障

系统故障是指造成系统停止运行的任何事件，使得系统需要重新启动。系统故障只能丢失数据缓冲区中的内容，影响正在执行的所有事务，但不会破坏数据库。系统故障中止了事务的执行过程，破坏了事务的原子性。由于缓冲区中的内容可能已部分写入数据库，系统重启后数据库可能处于不一致状态。

3) 介质故障

介质故障是指数据库的存储介质发生故障，如磁盘损坏和瞬间强磁场干扰等。这种故障直接破坏了数据库，会影响到所有正在读取这部分数据的事务。

2. 恢复的实现技术

为使数据库在发生故障后能够恢复，必须建立冗余数据。在故障发生后利用这些冗余数据实现数据库恢复。

建立冗余数据常用的技术是数据转储和建立日志文件。在一个数据库系统中，这两种方法一般是同时采用的。

数据转储是将数据库复制到另一个磁盘或磁带上保存起来的过程，又称为数据备份。数据转储又分为静态转储和动态转储。静态转储是指在转储期间不允许对数据库进行任何存取和修改活动，得到与数据库完全一致的副本。动态转储是指在转储期间允许对数据库进行存取和修改，即转储可以与事务并行执行。

静态转储又可分为完全转储和增量转储。完全转储是转储当前数据库的全部数据，增量转储只转储上次备份后发生改变的数据。

数据转储可以由数据库管理员 (DBA) 来操作。如静态转储，可以设定时间计划由

DBMS 定时执行;可以在事务程序中增加功能实现动态转储;也可以通过硬件系统的冗余磁盘阵列来实现。

日志文件是用来记录对数据库系统的更新操作的文件。在运行过程中,系统把事务开始、事务结束以及对数据库的插入、删除和修改的每一次操作作为一条记录写入日志文件中。每条记录包括的主要内容有执行操作的事务标识、操作类型、更新前数据的旧值(插入操作此项为空)、更新后的数据值(删除操作此项为空)、更新日期和更新时间。

为保证数据库是可恢复的,日志文件的登记必须遵循两条原则:

- 登记的次序严格按并发事务执行的时间次序;
- 必须先写日志文件,后写数据库。

3. 恢复策略

有了数据转储和日志文件,就可以在系统发生故障时进行恢复。

故障恢复一般有两个操作。

- 撤销事务(UNDO):将未完成的事务撤销,使数据库回复到事务执行前的正确状态。

撤销事务的过程如下:反向扫描未完成的事务日志(由后向前扫描),查找事务的更新操作;对该事务的更新操作执行逆操作,用日志文件记录中更新前的值写入数据库,把插入的记录从数据库中删除,把删除的记录重新插入数据库中;继续反向扫描日志文件,查找该事务的其他更新操作并执行操作直至事务开始标识处。

- 重做事务(REDO):对已经提交的事务重新执行。

重做事务的过程如下:从事务的开始标识起,正向扫描日志文件,重新执行日志文件登记的该事务对数据库的所有操作,直至事务结束标识处。

对于不同的故障,采取不同的恢复策略。

1) 事务故障的恢复

事务故障是事务在运行至正常终止点(SUMMIT 或 ROLLBACK)前中止,日志文件只有该事务的开始标识而没有结束标识。这类故障的恢复可通过撤销(UNDO)产生故障的事务,使数据库恢复到该事务执行前的正确状态来完成。

事务故障的恢复由系统自动完成,对用户是透明的。

2) 系统故障的恢复

系统故障会使数据库的数据不一致,原因有两个:一是未完成的事务对数据库的更新可能已写入数据库,二是已提交的事务对数据库的更新可能还在缓冲区中还没来得及写入数据库。因此,恢复操作就是要撤销故障发生时未完成的事务,重做(REDO)已提交的事务。

系统故障的恢复是在系统重启之后自动执行的。

3) 介质故障的恢复

介质故障会使数据库遭到破坏,需要重装数据库,装载故障前最近一次的备份和故障前的日志文件副本,再按照系统故障的恢复过程执行撤销和重做处理。

介质故障需要数据库管理员(DBA)参与,装入数据库的副本和日记文件的副本,再由系统执行撤销和重做操作。

11.9.3 数据库的安全性

数据库系统的安全性是指保护数据库,以防止不合法的使用所造成的数据泄露、更改或破坏。对数据库不合法的使用称为数据库的滥用。数据库的滥用可分为无意的和恶意的两种。无意的滥用在事务处理时容易发生系统故障和异常现象以及违反数据完整性约束等逻辑错误。恶意的滥用主要是指未经授权的读取(偷窃信息)和未经授权的修改操作(破坏数据)。

数据库的完整性是指尽可能避免对数据库的无意滥用;数据库的安全性是指尽可能地避免对数据库的恶意滥用。无意滥用可以通过约束来避免,而完全避免恶意滥用是不可能的,但可以尽量增加一些保护措施,提高数据库的安全性。

安全性措施可以从物理环境、网络环境、操作系统和数据库系统几个层次来实现,这里只讨论数据库系统层次上的安全性措施。

数据库系统的安全措施包括以下几个方面。

1. 权限机制

通过权限机制,限定用户对数据的操作权限,把数据的操作限定在具有指定权限的用户范围内。在标准 SQL 中定义的授权语句 GRANT 可用于实现权限管理。

2. 视图机制

通过建立用户视图(用户或应用程序只能通过视图来操作数据),保证了视图之外的数据的安全性。

3. 数据加密

对数据库中的数据进行加密,可以防止数据在存储和传输过程中失密。

11.9.4 数据库的完整性

数据库的完整性是指数据的正确性和相容性。如学生的性别只能是男或女,百分制的成绩只能取 0~100 的整数值等。为防止错误数据进入数据库,DBMS 提供了完整性约束机制。通过对数据库表结构进行约束,当用户修改数据时,由系统对数据修改进行完整性检查,将错误数据拒绝于数据库之外。

完整性约束条件作用的对象可以是表、行和列 3 种。列约束主要是对列的类型、取值

范围、精度、非空值及值不可重复等的约束条件。行约束是记录字段之间联系的约束条件,如余额应该等于存入金额减去支出金额的差值。表约束是表的主码约束、表与表间的参照完整性约束以及表中记录间的联系约束,如部门最高工资不能大于本部门平均工资的 5 倍。

列级约束、主码约束以及参照完整性约束是在数据库定义过程中定义的,并和数据库定义的其他信息存储在数据字典中。标准 SQL 的 DDL 语言提供了这种功能,其他相对复杂的约束需要编写触发器(trigger)程序实现。

在事务程序对数据库进行修改时,对于数据库定义的约束,由 DBMS 提供的完整性约束机制来检查,如果不符合约束条件则拒绝修改并给出提示。对于触发器程序的约束,由触发器机制执行程序来实现。

11.9.5 数据库的并发控制

数据库是一个共享资源,要供多个用户使用。如果事务程序一个一个地串行执行,一个事务必须等待正在执行的事务结束后才能被执行,这样就会造成系统资源的浪费,对于访问密集型的数据库系统,还会严重影响系统的响应速度,降低系统的性能。解决的办法就是使多个事务并行执行,但这种并行如果不加以限制,就会得到错误的数据甚至破坏数据库的一致性。

1. 并发操作

并发操作带来的数据不一致性有丢失修改、不可重复读和读脏数据 3 类,如图 11-2 所示。

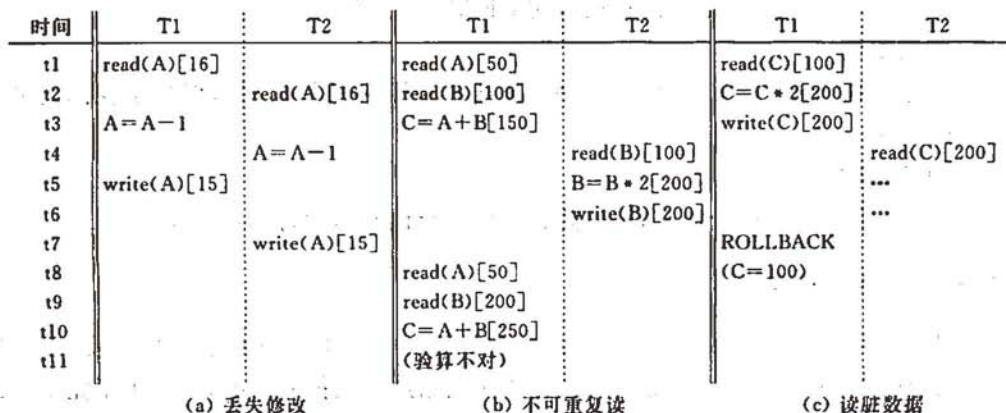


图 11-2 3 种数据不一致性

1) 丢失修改

如图 11-2(a)所示,事务 T1、T2 都是对数据 A 做减 1 操作。事务 T1 在时刻 t6 把 A 修改后的值 15 写入数据库,但事务 T2 在时刻 t7 再把它对 A 减 1 后的值 15 写入。两个事务都是对 A 的值进行减 1 操作并且都执行成功,但 A 中的值却只减了 1。如售票系统,实际上已同时售出了两张票,但数据库里的存票却只减了一张,造成数据的不一致。原因在于 T1 事务对数据库的修改被 T2 事务覆盖而丢失了,破坏了事务的隔离性。

2) 不可重复读

事务 T1 读取 A、B 的值后进行运算,事务 T2 在 t6 时刻对 B 的值做了修改以后,事务 T1 又重新读取 A、B 的值再运算,同一事务内对同一组数据的相同运算结果不同,显然与事实不相符。这同样是因为事务 T2 干扰了事务 T1 的独立性。

3) 读脏数据

事务 T1 对数据 C 修改之后,在 t4 时刻事务 T2 读取修改后的 C 值做处理,之后事务 T1 回滚,数据 C 恢复了原来的值,事务 T2 对 C 所做的处理是无效的,它读的是被丢掉的垃圾值。

通过以上 3 个例子,说明了在事务并行处理的过程中,因为多个事务对相同数据的访问,干扰了其他事务的处理,产生了数据的不一致性。这是违反事务隔离性而破坏了一致性。

解决问题的方法是从保证事务的隔离性入手。问题的焦点是事务在读写数据时不加控制而相互干扰。

2. 加锁

通过上面的例子可以知道,并发事务如果在数据读写时不加以控制,会破坏事务的隔离性和一致性。控制的手段就是加锁,在事务执行时限制其他事务对数据的读取。在并发控制中引入排他锁(exclusive locks,简称 X 锁)和共享锁(share locks,简称 S 锁)。

(1) 排他锁又称为写锁,用于数据写操作时进行锁定。如果事务 T 对数据 A 加上 X 锁,就只允许事务 T 读取和修改数据 A,其他事务不能对数据 A 再加任何锁,从而也不能读取和修改数据 A,直到事务 T 释放 A 上的锁。

(2) 共享锁又称为读锁,用于数据读操作时进行锁定。如果事务 T 对数据 A 加上了 S 锁,事务 T 就只能读数据 A 但不可以修改,其他事务可以再对数据 A 加 S 锁执行读取操作,只要数据 A 上有 S 锁,任何事务都只能再对其加 S 锁读取而不能加 X 锁修改。

3. 封锁协议

通过对数据加锁,可以限制其他事务对数据的访问,但这会降低事务的并发性。如何在保证事务一致性的前提下尽可能地提高并发性,这需要采用封锁协议来解决。

封锁协议是对数据加锁类型、加锁时间和释放锁时间的一些规则的描述。

1) 一级封锁协议

一级封锁协议：事务 T 在修改数据 A 之前必须先对其加 X 锁，直到事务结束才释放 X 锁。

一级封锁协议使得一个事务在修改数据期间，其他事务不能对该数据进行修改（直至事务结束），解决了丢失修改的问题。

2) 二级封锁协议

二级封锁协议：一级封锁协议加上事务 T 在读取数据 A 之前必须对其加上 S 锁，读完后即可释放 S 锁。

二级封锁协议使得一个事务不能读取其他事务修改中的数据。解决了读脏数据的问题。但是，如果事务 T 在读取数据 A 之后，其他事务再对 A 做完修改，事务 T 再读取 A，还会产生不可重复读的错误。

3) 三级封锁协议

三级封锁协议：一级封锁协议加上事务 T 在读取数据 A 之前必须对其加上 S 锁，直到事务结束才释放 S 锁。

三级封锁协议使得一个事务在读取数据期间，其他事务只能读取该数据而不能修改，解决了不可重复读的问题。

4) 两段锁协议

两段锁协议的内容是：对任何数据进行读写之前必须对该数据加锁。在释放一个封锁之后，事务不再申请和获得任何其他封锁。

两段锁协议缩短了持锁时间，提高了并发性，同时解决了数据的不一致性。

11.10 小结

本章围绕数据库应用系统设计与开发，从需求分析入手，讲述了数据库设计开发各个阶段的设计方法。数据库保护方面的内容也是设计和开发应用程序所要求的。

数据库应用系统的设计与开发是一项复杂的系统工程。在遵循一般系统开发与设计方法的同时，又要考虑数据库设计方面的特点（它在程序开发方面与事务密切相关）。一个成功的数据库应用系统，是建立在好的系统分析与设计、具体数据库设计和数据库应用程序的基础上的。

第 12 章 数据库运行与管理

数据库系统投入使用后,会面临着数据的不断更新和频繁的访问,随着时间的推移,会出现各种情况影响数据库的效能和稳定。如何保证数据库安全稳定地运行,针对运行中出现地各种问题如何解决,如何调整才能使数据库系统发挥更大地效能,是本章要讨论的问题。

12.1 数据库系统的运行计划

为保证数据库系统安全稳定地运行,需要综合考虑可能遇到的各种问题,制定详尽的运行计划和应对措施。任何因素导致系统出现问题,都可能给企业带来损失。

12.1.1 运行策略的确定

要使数据库系统能够正常运行,必须制定运行策略。运行策略的制定要考虑正常运行策略和非正常运行策略两个方面。

正常运行策略是指在正常运行状态下的数据库执行策略。任何一个系统在一般情况下都有相对固定的用户群和访问量,系统的负载相对稳定。

需要从以下几个方面考虑正常运行策略。

1) 系统运行对物理环境的要求

为保障系统的稳定运行,离不开系统的物理环境保障。物理环境包括运行场地的温度、湿度、通风条件、灰尘指标和电力供应等外部条件。

2) 系统运行对人员的要求

企业中的数据库运行需要专人服务。应成立数据库运行管理机构,专门负责数据库系统的运行。

3) 数据库的安全性策略

数据库的运行离不开用户的访问和操作,安全性策略包括网络安全、用户的权限管理、设备的安全以及数据的安全等方面。

4) 数据库备份和恢复策略

在数据库系统运行中数据是不断变更和增长的。有些系统会产生大量的数据,这些数据如果不能及时从系统中导出,系统的存储设备很快会被占而不能正常运行。因此需

要根据业务量,制定数据备份策略,定期从系统中导出数据。同时,备份也是系统故障恢复所必需的。

非正常运行策略是指在特殊时期的数据库运行策略。系统运行不可能一成不变,在各种因素的影响下,系统会处于特殊的运行时期。非正常运行策略需要从以下几个方面考虑:

(1) 突发事件的应对策略。

突发事件可能是突然断电、设备故障等因素,甚至可能是火灾、水灾等人力不可抗拒的自然灾害,必须要有及时的应对策略;如启动备用电源和备用设备,使系统能够正常运行。

(2) 高负载状态的应对策略。

数据库系统的高负载状态与企业的业务相关,有些是可以预计的,如节日中的话务系统,有些则是事先难以估计的,如大幅涨跌时的股票交易系统。针对高负载时的系统运行,也要求有正确的应对策略,进行系统负载平衡。

12.1.2 确定数据库系统监控对象和监控方式

在数据库系统运行过程中,管理员需要及时了解数据库的运行状态,掌握运行状态中的各种指标,为改进系统提供依据。对系统运行状态的了解可采用监控手段。

数据库系统监控的对象分别是系统性能、系统故障和系统安全,依照监控对象的不同,系统监控分为性能监控、故障监控、安全监控。

性能监控是掌握系统运行性能的手段。性能监控应当从资源占用率、事务响应时间、事务量、死锁和用户量等方面考虑。

故障监控是保障数据库系统正常运行的手段。从数据库系统故障的类型入手,监控事务故障、系统故障和介质故障。当出现需要管理员干预的故障时应及时恢复。

安全监控是对数据库安全事件的监控,包括入侵监控、用户访问监控和病毒监控等。

在进行系统监控的同时,可以设定出现严重问题时的系统报警,及时通知管理员进行干预,保障系统稳定地运行。

数据库系统的监控方式分为系统监控和应用程序监控。

系统监控可通过 DBMS 提供的监控功能,设定参数后,由系统自动监控。不同的 DBMS 软件都不同程度地提供了监控功能,管理员可以有效地利用。

应用程序监控需要管理人员根据具体情况编制应用程序进行系统监控,是对 DBMS 监控功能的补充。

系统日志是监控的主要依据。日志文件详细记录了系统运行中的各种信息,管理员可以从日志文件中了解系统运行状态和事件,以此为据发现系统运行中的问题。

12.1.3 数据库系统管理计划

数据库系统运行过程中离不开对系统的有效管理,包括性能管理、故障/恢复管理、安全性管理、完整性管理、用户教育与培训以及系统的维护,有关内容可以参阅本章数据库管理一节。

12.2 数据库系统的运行和维护

12.2.1 监控数据的收集与分析

系统监控需要动态地掌握数据库的运行状态。监控过程中对系统运行信息的记录,称为监控数据。监控数据是发现系统问题和改进系统性能的依据。依照监控的类型,监控数据分为性能监控数据、故障监控数据和安全监控数据。

监控数据通常可以从 DBMS 系统监控功能指定的记录文件中获取。有些运行信息可能记录在系统日志文件中,管理员编制的监控脚本也可以指定监控数据的存储文件,从这些文件中可以得到监控数据。

监控数据是系统运行状态的反映,分析监控数据,目的在于判定系统运行是否正常,是否满足设计要求和应用要求,出现问题的根源在哪里,并给出解决问题的方案,为进一步改进系统提供依据。

性能监控数据包括磁盘使用信息(碎片量、剩余空间和日志文件增长情况),I/O 操作数量、频度及响应时间,缓冲区命中率,事务量及锁状况。通过分析这些数据,找出影响性能的问题所在,为下一步的性能调整提供依据。

通过对故障监控数据的分析,可以找出故障的原因。确定是事务处理程序的内部错误,还是系统调度的问题,以及是否系统硬件故障,做出相应的处理。

安全监控数据主要是用户访问和修改数据库的记录。可用于判定是否有未授权用户的存取,分析安全漏洞的原因,以便对用户管理和应用程序加以改进。

12.2.2 稳定运行中的业务持续性

业务持续性是指一个组织的主要业务流程、营运服务以及 IT 服务能够得到连续的处理。

在一个突发事件中,公司的主要业务、服务流程、设备和人员等因素都有着各自的持续性要求。公司的 IT 部门和其他职能部门必须相互配合,这种配合不仅仅体现在业务持续的计划中,更需要在具体的实施过程中得到实现。

业务持续性需要从以下方面考虑:

(1) 界定哪些是不允许停工的持续性业务,哪些是允许有一定时间停工期的弹性业务。

(2) 要有业务持续性的技术体系,如高效率服务器、存储系统、网络 and DBMS。

(3) 检测和响应管理,包括紧急决策的制定、准备工作、最初的紧急响应和系统恢复的详细程序等。

(4) 要有保障业务持续性的设备。

(5) 界定相关人员的职务和权责。包括各类技术人员(程序员、管理员和操作员)、执行经理(紧急事件决策者)、设备(电力、供冷和电缆)管理人员、人力资源(人事问题和需求)、业务实体(业务流程)以及外部组织(外包机构、电信和供应商等)。

12.2.3 数据库维护

在数据库系统的运行过程中,可能会由于某些原因需要修改数据库的结构,称为数据库重构。重构包括表结构的修改和视图的修改。

表结构的修改有数据列的增删与修改、约束的修改以及表的分解与合并。由于DBMS具有一定的逻辑独立性,这些修改可能不需要修改应用程序。具体说明如下:

(1) 修改表中的属性列名或数据类型,必须修改使用该表的应用程序。因此应尽量减少做这样的修改。

(2) 增加和删除属性只须修改使用该列的应用程序。

(3) 修改的约束如果是DBMS支持的约束,如主码约束、参照完整性约束和检查约束,一般不需要修改应用程序。复杂的约束可以通过修改触发器程序实现。

(4) 表的分解可以通过建立与分解前的表同名的视图来避免修改应用程序。但这样会引起性能的下降,如果分解的目的是为了提高性能,则需要修改应用程序,使之只访问分解后的一个表。

(5) 表的合并通常也是为了提高系统性能,可以通过建立两个与原表同名的视图来避免修改应用程序。

视图机制一方面可以实现数据的逻辑独立性,另一方面可以实现数据的安全性,将不允许应用程序访问的数据屏蔽在视图之外。在数据库重构过程中引入或修改视图,可能会影响数据的安全性,因此必须对视图进行评价和验证,保证不会因为数据库的重构而引起数据的泄密。

文档是对系统结构和实现的描述,在系统的设计、开发和维护过程中起着重要的指导作用。文档必须与系统保持高度的一致性,否则会造成人为的困难和错误,甚至危及系统的生命。数据库重构过程中的所有修改,必须在文档中体现出来。

12.2.4 数据库系统的运行统计

系统监控和系统运行统计是 DBA 掌握数据库系统运行状态最有效的手段。系统监控通常用来保障系统的稳定运行,运行统计则用来了解系统性能,作为性能调整的依据。

系统的运行统计是通过 DBMS 提供的工具实现的,也有第三方软件可供使用。可以将统计数据以图和表等多种形式提供,并给出相应的分析结果。DBA 可以通过统计数据,了解系统性能和资源占用情况,实施系统改进和资源配置,以提高系统性能。

运行统计可以是长期的,也可以是阶段性的。如对访问量的统计是长期的,峰值时期的统计则是为了掌握系统的负荷能力,因此是阶段性的。

12.2.5 数据库系统的审计

审计是一种 DBMS 工具,它记录数据库资源和权限的使用情况。启用审计功能,可以产生审计跟踪信息,包括哪些数据库对象受到了影响,谁在什么时候执行了这些操作。

审计是被动的,它只能跟踪对数据库的修改而不能防止。但作为一个安全性手段,能起到对非法入侵的威慑作用,可以据此追究非法入侵者的法律责任。

审计功能的开启会影响到系统的性能,尤其是在一个忙碌的系统中,会导致性能的降低。而且审计跟踪信息保存会引起存储空间的问题。解决这一问题的方法是在 DBMS 范围内的不同级别上进行审计操作,例如,在数据库级别、数据库对象级别或用户级别上进行审计。根据不同级别有选择地进行审计,可以使对存储和性能的负面影响降到最小。

12.3 数据库系统的管理

12.3.1 数据字典的管理

数据字典(data dictionary)是存储在数据库中的所有对象信息的知识库,存有用户及其权限信息、所有数据对象、表的约束条件和统计分析数据库的视图等信息。通常把数据字典中存储的数据称为元数据(metadata)。元数据用来描述数据,如字段的类型和长度等。系统对数据库的访问是根据数据字典中的元数据所提供的信息来访问具体数据的,同时也可以通过元数据了解数据库对象的信息。

当用户使用 DDL 语言定义数据库对象或某些 DML 语言进行表扩展等操作时,系统会自动修改数据字典中的元数据。数据字典是只读的,不同的 DBMS 都提供相应的数据字典访问命令,可通过这些命令访问数据字典的内容。

在系统运行阶段,对数据库对象结构信息的修改会自动地反映到数据字典中。这些

修改可能会关系到应用程序对数据库的正确访问。也可以通过数据字典的信息对应用程序做相应的修改。

12.3.2 数据完整性维护和管理

数据库的完整性是指数据语言上的一致性,从语义角度限定数据。有关完整性的详细说明参见本书第 11 章的内容。

数据的完整性是通过 DBMS 系统提供的完整性约束机制和应用程序来实现的,以保证运行过程中数据的正确性。

在系统运行过程中,对数据完整性的维护和管理采用两种方式:

(1) 对于 DBMS 管理的约束,可通过修改数据库的定义,如增加或删除实体完整性约束、参照完整性约束以及检查约束来实现。

(2) 对于应用程序实现的复杂的完整性约束,可通过分析和修改应用程序(通常是触发器程序)来实现。

12.3.3 数据库的存储管理

数据库中的数据是以文件的形式存储在物理存储设备(通常是磁盘系统)上的。应用程序通过 DBMS 完成 I/O 操作来访问数据。I/O 操作的效率直接影响到系统的运行效率,因此提高系统访问效率的有效手段就是提高 I/O 操作的效率。

在数据库系统运行过程中,随着数据的不断变更,会影响到系统的响应效率。通过以下手段进行存储管理,可有效地提高系统性能:

(1) 索引文件和数据文件分开存储,事务日志文件存储在高速设备上;

(2) 适时修改数据文件和索引文件的页面大小;

(3) 定期对数据进行排序;

(4) 增加必要的索引项。

除进行数据库的存储管理之外,也可以通过增加计算机内存,引入高速存储设备等外部方式提高系统的访问效率。

12.3.4 备份和恢复

在数据库系统运行过程中,可能会发生故障而破坏数据。预防措施是进行数据库备份。有关备份与恢复的概念参阅本书第 11 章。

随着存储设备稳定性的不断提高,硬件故障发生的概率越来越小,故障主要集中在由应用程序引起的事务故障和系统故障上。目前绝大多数 DBMS 系统都提供备份和恢复机制。在系统运行过程中,管理员需要做的工作主要是做好备份和日志管理

工作。

针对备份计划的制定和实施,有以下建议:

(1) 根据数据变更情况,设定合理的备份周期和备份时间,最好是在业务量最小的时段进行备份;

(2) 把事务日志文件保存在最稳定的存储设备上;

(3) 定期在事务日志文件中加入检查点(checkpoint)。

检查点记录了数据库的正确状态点,在数据库恢复过程中,可以反向扫描日志文件,找到第一个检查点,执行 Undo 和 Redo 操作,减少恢复的时间开销。

12.3.5 并发控制与死锁管理

多用户数据库管理系统一般都提供并发控制机制,来实现事务的并发调度并进行死锁管理。在实际运行过程中,死锁的产生往往是因为事务程序的错误引起的。管理员通过系统监控工具或系统日志,可以找出频繁产生死锁的事务。分析死锁的原因,修改事务程序来减少死锁,提高系统的并发性。

12.3.6 数据安全性管理

有关数据库的安全性知识可以参阅本书第 11 章的相关内容。在实际运行中可以从以下几个方面实现安全性管理:

(1) 建立网络级安全,重点是防火墙的设置;

(2) 操作系统级安全,进行登录用户的管理;

(3) DBMS 级安全,对访问数据库的用户进行密码验证;

(4) 角色和用户的授权管理;

(5) 建立视图和存储过程以加强安全性;

(6) 使用审计功能,为追究非法入侵者的法律责任提供证据,发现安全漏洞。

12.4 性能调整

在数据库系统的运行过程中,尽可能地提高系统的性能,是管理员的主要工作之一。系统的性能一方面取决于 DBMS 的性能及其参数设定,而在一定的 DBMS 环境下,与具体的应用系统也有很大的关系,可以通过调整来提高性能。

12.4.1 SQL 语句的编码检验

通过 DBMS 提供的监控和统计功能,找出频繁执行的 SQL 语句(通常是查询语句),

对其进行优化。常用的策略如下：

- (1) 尽可能地减少多表查询或建立物化视图；
- (2) 以不相关子查询替代相关子查询；
- (3) 只检索需要的列；
- (4) 用带 IN 的条件子句等价替换 OR 子句；
- (5) 经常提交 COMMIT，以尽早释放锁。

12.4.2 表设计的评价

在设计阶段，关系模式的设计应当符合 3NF 或 BCNF，目的是为了减少数据冗余和消除操作异常。在数据库系统运行过程中，需要根据实际情况对表进行调整。调整的原则是：

- (1) 如果频繁地访问涉及的是对两个相关的表进行连接操作，则考虑将其合并；
- (2) 如果频繁地访问只是在表中的某一部分字段上进行，则考虑分解表，将该部分单独作为一个表；
- (3) 对于很少更新的表，引入物化视图。

12.4.3 索引改进

改进系统中的索引可以提高性能。可以针对于具体的情况，适当地调整索引。调整的原则是：

- (1) 如果查询是瓶颈，则在关系上建立适当的索引。通常，在作为查询条件的属性上建立索引可以提高查询效率。
- (2) 如果更新是瓶颈，因每次更新都会重建表上的索引，引起效率的降低，则考虑删除某些索引。
- (3) 选择适当的索引类型。如果经常使用范围查询，则 B 树索引比散列索引更高效。
- (4) 将有利于大多数数据查询和更新的索引设为聚簇索引。

12.4.4 设备增强

在系统运行过程中，如果经过各种调整后仍不能满足性能要求，则应当考虑增强系统设备。

- (1) 引入高速的计算机；
- (2) 增加系统内存；

(3) 使用高速的网络设备；

(4) 使用高速的存储设备。

设备的增强需要企业的资金投入,应当考虑合适的性价比和投入产出比,还需要说服决策者同意。

12.5 用户支持

12.5.1 用户培训

数据库应用系统离不开用户的使用,良好的界面设计和用户手册可以方便用户的使用,而定期的培训也是必不可少的。

用户的分类:根据使用系统的内容和权限,可以将用户分为以下 3 类:

(1) 各级管理者 通常是业务经理直至总经理等管理层,他们主要关心系统统计数据。

(2) 业务雇员 这类人员是与系统打交道最多的人。

(3) 外部用户 对于开放的数据库系统,可能会有企业之外的人员访问甚至修改数据库的内容。如电子商务、网上银行中的客户。

用户培训主要针对内部人员,即管理者、雇员尤其是业务雇员。由于这类人员往往变动性很大,对业务较熟悉,计算机操作水平有限,必须经过培训考核才能胜任。

培训的内容和目的:

(1) 了解业务流程及规范,熟悉数据库系统的使用。

(2) 掌握应用程序的操作,正确地使用和维护数据。

(3) 培养安全意识,防止泄露或破坏数据。

12.5.2 售后服务

由于数据库应用系统的复杂性,无论是 DBMS 的供应商还是应用系统的开发商,都不可能保证自己的产品没有任何问题,最终用户也不可能完全有能力解决系统中出现的问题,良好的售后服务直接关系到企业的信誉。

售后服务通常包含的内容如下:

(1) 成立专门的客户服务机构,解决用户的技术问题,包括热线服务和上门服务。

(2) 用户技术培训。

(3) 优惠的系统升级。

12.6 小结

本章从应用的角度讲述了数据库系统运行期间的管理和维护工作,从 DBA 的角度描述了系统运行中的各项工作。实际中运行的不同厂商的 DBMS 有多种,具体的实施步骤需要参照 DBMS 的文档执行,但这些方法是通用的。作为一名合格的 DBA,需要有深厚广博的计算机软、硬件方面的知识,尤其是对 DBMS 内部的技术有很好的了解。还要有分析问题和解决问题的能力,需要不断地学习。

第 13 章 网络与数据库

随着网络技术的迅速发展,人们可以通过网络访问异地资源,实现在地域上分散情况下的数据传输,达到数据共享的目的。但是,如果网络的数据传输仅以文件系统为基础,那么就很难对数据进行有效管理,数据冗余、数据一致性问题就得不到解决。在这种情况下,人们自然就会将网络技术与数据库技术相结合。另外,随着数据量的不断增大,要求数据分布在不同的地域,那么传统的、集中式数据库系统很难满足这一需求,因此,数据库技术和网络技术相结合就成为必然。

网络技术与数据库技术相结合会产生不同的结果:单一的、集中式的数据库在网络环境下的应用、网络连接的分散数据库以及分布式数据库等。数据库的作用就是为了统一、集中地管理数据,并负责给应用程序提供数据。例如,应用程序在一台计算机上运行,数据库系统在另外一台计算机上运行,两台计算机通过网络连接。当应用程序需要数据时,通过网络向数据库系统发出请求,数据库系统查询数据后再通过网络将结果返回应用程序,应用程序即可对数据进行操作了。这就是一个典型的单一的、集中式的数据库系统在网络环境下的应用,许多大型的应用系统都采用这样一种体系结构,即 C/S(client-server, 客户/服务器)体系结构。可以看出,即使是简单的 C/S 体系结构,一般也要通过网络将客户端和服务端连接起来。

进一步讲,如果后台的数据库系统不止一个,并且数据库系统之间也采用网络连接起来,那么此时系统就成为网络连接的分散数据库系统了。在这种情况下,应用程序要协调各个分散的数据库系统,并且应用程序要确切地知道数据在哪一个数据库系统中。

对应用来说,协调各个分散的数据库通常是很困难的,所以人们希望数据库管理系统能够通过网络协调分散在各个地域的数据库,而应用程序只关心应用逻辑的处理,数据的物理位置对应用户来说是透明的。简单地说,借助网络把分散在不同地域的数据管理起来,如同集中式数据库系统管理本地数据一样。这是分布式数据库系统要实现的目标之一,同时,分布式数据库还必须满足各个地域的数据库系统的自治性,也就是各个地域的数据库系统能够支持本地的应用。

13.1 分布式数据库

分布式数据库系统是数据库系统和计算机网络相结合的产物。一方面,由于计算机功能增强,成本下降,几乎每个办公室、实验室及个人用户都可以拥有自己的计算机,从而

增加了数据分散处理的需求。另一方面,由于通信技术的迅速发展,出现了各种计算机网络,降低了数据传输费用。而计算机局部网络的广泛应用,则为分布式数据库系统的出现提供了实现的可能性。

分布式数据库系统的研制,经历了一个艰难而曲折的过程。最初由 IBM, UNIVAC 等设计单位提出在 APAC 和 CYCLADES 计划中发展起来的技术,只实现了分布式系统的信息通信,而分布式环境中的信息存贮、分配、交换以及处理等问题均没有得到妥善的解决。这说明在集中式数据库系统中所使用的传统技术,对于解决上述问题已不再适用。因此,简单的移植是不够的,必须根据分布式系统的特殊要求,对重新考虑实现技术。

自 1975 年以来,美国、法国、德国等国家的许多大学和科研单位都在研究分布式数据库领域里的各种问题,力图实现分布式数据库系统。随着各种理论问题研究的进展,以及人们对分布式数据库系统认识的深化,一些国家的政府也先后拟订了发展这一领域的计划,如美国的 SDD-1 计划、法国的 CIRIUS 计划以及德国的 POREL 计划等。分布式数据库系统中最著名的有柏林工业大学的 VDN 系统和斯图加特大学的 POREL 系统等。

13.1.1 分布式数据库的概念

1. 分布式数据库的定义

分布式数据库系统(distributed database system, DDBS)是针对地理上分散,而管理上又需要不同程度集中管理的需求而提出的一种数据管理信息系统。在明确给出分布式数据库的定义之前,先介绍一下一般的分布式数据库系统的组成,参见图 13-1。

可以看出,分布式数据库系统首先是由多个不同节点或场地的数据库系统通过网络连接而成的(如不加特别说明,本章中的场地和节点表示同一含义),每个节点都有各自的数据库管理系统(local database management system, LDBMS),同时还有全局数据库管理系统(global database management system, DDBMS)。图中的局部用户是针对某一个节点而言的,局部用户只关心他所访问的节点上的数据,而全局用户则可能需要访问多个节点上的数据。每个节点的数据库管理系统(LDBMS)响应局部用户的应用请求,全局数据库管理系统(GDBMS)则为全局用户提供服务,全局用户可以从任意一个节点访问分布式数据库系统中的数据。

在一个计算机网络中,每个节点都装有数据库系统,节点数据虽然达到共享,但如果没有统一的管理,对于用户来说,使用数据库数据时必须指明数据库所在的场地,无法实现场地的透明性,就达不到分布式数据库的目标。如果只在计算机网络中某一场地设置数据库系统,其他场地不设数据库系统,而是有多个终端(远程)用户,显然又达不到数据分散存储的目的,所以分布式数据库至少应该有场地透明性和分散存储两个特点。

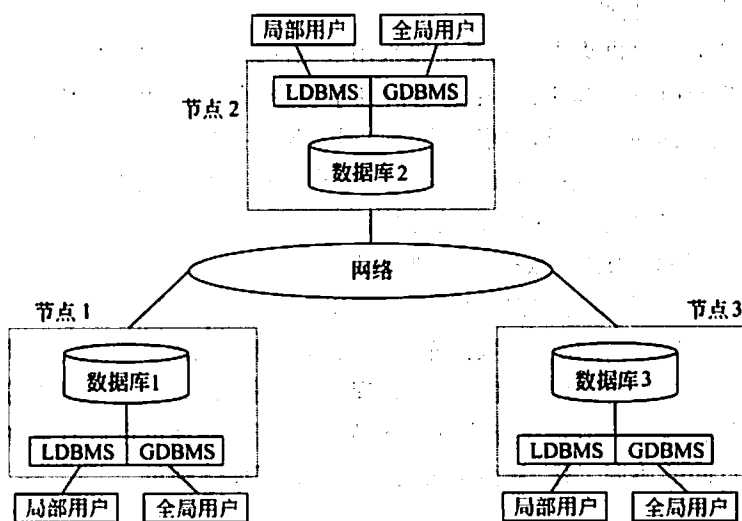


图 13-1 分布式数据库系统组成图

另外,作为一个整体,分布式数据库系统应该保证数据的一致性,这就意味着分布式数据库系统中的各个局部数据库之间应该具有逻辑相关性。根据上述特点,这里给出分布式数据库的定义。

满足下面条件的数据库系统被称为完全分布式数据库系统。

- (1) 分布性: 即数据存储在多个不同的节点上。
- (2) 逻辑相关性: 即数据库系统内的数据在逻辑上具有相互关联的特性。
- (3) 场地透明性: 即使用分布式数据库中的数据时不需指明数据所在的位置。
- (4) 场地自治性: 即每一个单独的节点能够执行局部的应用请求。

分布式数据库系统的分布性可以让人们区分单一的集中式数据库与分布式数据库,而根据逻辑相关性,可以将分布式数据库与一组局部数据库或存储在计算机网络中不同节点的文件系统区分开来。场地的透明性和场地的自治性则可以多机处理系统或并行系统区分开来。

2. 分布式数据库的特点

分布式数据库系统是传统集中式数据库系统的发展,因此它具有集中式数据库系统的特点。同时,由于它的分布性而又使这些特点具有新的含义。传统的数据库系统针对文件系统的弱点,采用了集中控制以实现数据共享,这是其最主要的特色。对于分布式数据库系统来说,由于数据的分散性,分布式数据库系统具有分散与集中统一的特性。下面给出了分布式数据库的几个主要的特点。

1) 数据的集中控制性

能够对信息资源提供集中控制是主张采用数据库最强有力的动机之一。数据库是随着信息系统的演变而发展起来的,在这些信息系统中,每个应用程序都有自己的专用文件,这样就不利于数据的管理和共享。由于数据本身已被当作企业的重要资源,在这样的需求推动下,传统的数据库系统应运而生。分布式数据库系统是在传统数据库系统的基础上的发展,所以,它也具有集中控制的特性。

在传统的数据库系统中,数据库管理员(Database Administrator, DBA)的基本任务是保证数据的安全,并负责对数据进行管理,使用户和应用能够高效地访问数据。而在分布式数据库中,可以认为存在全局数据库管理员和局部数据库管理员。这是一种分层控制结构,一般来说,全局数据库管理员负责管理所有数据库,而局部数据库管理员只负责各自节点的局部数据库。但是在有些情况下,局部数据库管理员可以有更高的自主性,甚至能够完成节点间的协调工作,从而不再需要全局数据库管理员。

2) 数据独立性

数据独立性也是集中式数据库相对于文件系统的一大特征。独立性指的是数据的组成对应用程序来说是透明的。应用程序只需要考虑数据的逻辑结构,而不用考虑数据的物理存放,因而数据在物理组织上的改变不会影响应用程序。

在分布式数据库系统中,数据的独立性同样具有重要的意义。分布式数据库的数据独立性除了具有传统意义上数据独立性的含义,还有分布式透明的含义。所谓分布式透明是指:虽然应用程序面对的是分散存放的数据,但就像使用集中式数据库一样,不必考虑数据库的分布特性。

3) 数据冗余可控性

将数据组织在数据库中可以方便地实现数据共享,因此要尽量减少数据冗余。这不仅能够降低存储代价,还可提高查询效率,便于维护数据的一致性,这是数据库系统优于文件系统的特点之一。但对数据库系统来说,也不可能达到绝对无冗余数据。

对于分布式数据库来说,由于数据存储的分散性,需要在网络上传输数据。与集中式数据库相比,查询中就增加了传输代价。因此,分布式数据库中的数据一般存储在经常使用的场地上,但应用对两个或两个以上场地的同一数据有存取要求也是时常发生的,而且当传输代价高于存储代价时,可以将同一数据存储两个(甚至更多)场地上,以节省传输的开销。另外,数据有多个副本,也可以提高系统的可用性,即当系统中某个节点发生故障时,因为数据有其他副本在非故障场地上,数据仍然是可用的,从而保证了数据的完备性。由于这种冗余度是在系统控制下的,所以给系统造成的不利影响是可控制的。

另外,由于可用副本的存在,也相应地提高了场地自治性的性能。

4) 场地自治性

在分布式数据库系统中,多个场地的局部数据库逻辑上为一个整体,这个整体称为全局数据库,并可以为分布式数据库系统的所有用户使用,这种应用称为分布式数据库的全局应用,其用户为全局用户。同时,分布式数据库系统还允许用户只使用本地的局部数据库,这种应用为局部应用,其用户为局部用户。局部用户所使用的数据甚至可以不参与到全局数据库中去,这种局部应用独立于全局应用的特性就是局部数据库的自治性。

由于自治性,对每个场地来说就有两种数据,一种是参与全局数据库的局部数据,另一种则是不参与全局数据库的数据。

5) 存取的有效性

在传统的数据库系统中,采用二次索引和文件链接等复杂的存储结构是提高存取效率的主要方法。但在分布式数据库系统中,仅仅采用复杂的存取结构并不是一个正确的方法。分布式数据库系统的全局查询可以分解成等效的子查询,即全局查询的执行计划可分解成多个子查询执行计划(它是根据系统的全局优化策略产生的),而子查询计划又是在各场地上分布执行的。因此,分布式数据库系统中的查询优化有两个级别:全局优化和局部优化。

全局优化主要决定在多个副本中选取合适的场地副本,使得场地间的数据传输量传输次数最少,从而使系统通信开销少。而局部优化就和传统的集中式数据库中的优化是一致的。

13.1.2 分布式数据库的体系结构

1. 分布式数据库的模式结构

分布式数据库的模式结构是目前国内外尚在讨论的问题,还没有统一的标准,这是因为对分布式数据库系统数据独立性要求的程度不同,其抽象层次也不同。我国在多年研究与开发分布式数据库及制定《分布式数据库系统标准》中,提出了把分布式数据库抽象为4层的模式结构,参见图13-2。这一模式结构得到了国内外一定程度的支持和认同。

这种4层模式划分为:全局外层、全局概念层、局部概念层和局部内层。在各层间还有相应的层间映射。4层模式的划分不仅适用于完全透明的分布式数据库系统,而且也适合各种透明性要求的分布式数据库系统。无论是对同构型分布式数据库系统,还是异构型分布式数据库系统都能适用。

1) 全局外层

分布式数据库是一组分布的局部物理数据库的逻辑集合。分布式数据库的全局视图如同集中式数据库一样,由多个用户视图组成。用户视图是针对分布式数据库特定的全局用户的,是对分布式数据库的最高层的抽象。

分布式数据库与集中式数据库的视图有同样的概念,不同的是,它不是从某一个具体

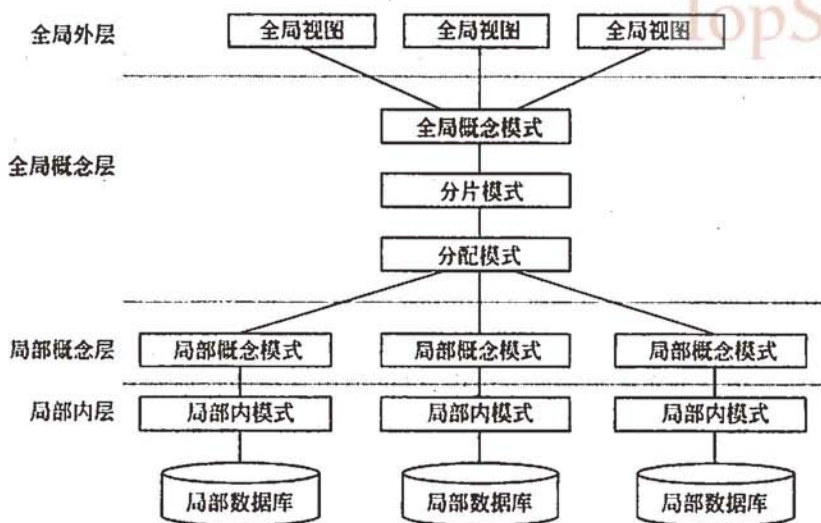


图 13-2 分布式数据库结构模式图

场地上的局部数据库中抽取，而是从一个虚拟的各局部数据库逻辑集合中抽取。对全局用户而言，不论他在分布式数据库系统中的哪一个节点上访问系统中的数据，都可以认为所有的数据库都在本地，而且他只须关心自己所使用的那部分数据。

如果是完全透明的关系模型的分布式数据库结构，则视图就和集中式数据库的视图一样，其定义方式也基本相同。因此全局用户在使用视图时，就不必关心数据的分片和具体的物理细节。若为非完全透明的分布式数据库，则在视图定义中，根据透明性支持的程度，需要给出一定的数据细节和物理存取细节等。

2) 全局概念层

全局概念层是分布式数据库的整体抽象，包含了系统中全部数据的特性和逻辑结构。就像集中式数据库中概念视图一样，是对数据库整体的描述。但在分布式数据库的 4 层抽象结构中，全局概念层比集中式的概念层有更多的描述。

从分布透明特性来说，分布式数据库的全局概念层应具有 3 种模式描述信息。

(1) 全局概念模式：描述分布式数据库全局数据的逻辑结构，是分布式数据库的全局概念视图。与集中式数据库的概念视图的定义相似，全局概念模式应包含模式名、属性名以及每种属性的数据类型的定义和长度。

(2) 分片模式：描述全局数据逻辑划分的视图，它是全局数据的逻辑结构根据某种条件的划分，每一个逻辑划分就是一个片段或称为分片。

(3) 分配模式：描述局部逻辑的局部物理结构是划分后的片段(或分片)的物理分配

视图。与集中式数据库物理存储结构的概念不同,这种分配模式是全局概念层的内容。

分布式数据库的定义语言除了需要提供概念模式的定义语句外,还必须提供分片模式和分配模式的定义语句。

从全局模式到分片模式,再到分配模式,它们之间存在着映射。全局概念模式到分片模式的映射是一对多的,即一个全局概念模式有若干个分片模式与之相对应,而一个分片模式只能对应一个全局概念模式。分片模式到分配模式的映射可以是一对多的或者一对一的,这是由数据分布的冗余策略所决定的。当采用一对多时,表明分片数据有多个副本存储在不同的场地上,且一般情况下同一场地不允许有相同的副本存在;采用一对一时,则表明数据是非冗余的,即分片数据只有一个副本。

从全局概念层观察,分布式数据库定义了全局数据的逻辑结构、逻辑分布性和物理分布性,但并不涉及全局数据在每个局部场地上的物理存储细节。所以全局概念层仍然是概念层视图,或全局数据库管理员视图。因此,全局数据库管理员将负责全局数据结构的定义、逻辑分布的定义和物理分布的定义。

分布式数据库的全局数据分布性描述对关系数据模型最为有利。对于关系型分布式数据库来说,全局概念模式是由一组全局关系模式的定义组成的,分片模式是对全局关系模式的逻辑划分定义,即由片段定义,或子关系模式定义组成,所以可以将片段看作是全局关系的逻辑组成,即逻辑片段;分配模式是对于子关系模式的描述,因此分配模式决定了子关系的物理场地,即决定子关系的物理片段。

3) 局部概念层

局部概念层由局部概念模式描述,一般情况下,它是全局概念模式的子集。全局概念模式经逻辑划分后被分配在各局部场地上。

在分布式数据库局部场地上,每个全局关系由该全局关系的若干个(可以是全部)逻辑片段对应的物理片段集合而成,该集合是一个全局关系在某个局部场地上的物理映像,全体物理映像组成局部概念模式。如果两个场地上的所有物理映像都相同,则其中一个场地上的数据必然是另一个场地的副本,两个场地的局部概念模式亦相同。

如果分布式数据库只支持全局应用,则局部概念模式可理解为局部数据库的概念模式和外模式。在此情况下,外模式和概念模式是相同的。如果分布式数据库还支持局部用户,而局部用户定义的局部数据不参与分布式数据库的全局数据,则局部概念层还应划分为局部外模式和局部模式,并且由局部 DBA 描述,这些将不属于全局概念模式。这里值得注意的是,全局数据和局部数据的管理分别由全局 DBA 和局部 DBA 管理。因此,全局用户是否可以使用全局数据应由全局 DBA 授权,局部 DBA 无权授予全局用户各种权限。

当全局数据模型与局部数据模型不同时,物理映像与各局部数据库的数据模型之间

还必须要有数据模型的转换。即使数据模型相同时,也可能有数据类型和格式的各种转换。也就是说,各局部数据库是多种数据模型构成的数据库时,在组成分布式数据库时,需要一个统一的全局描述,而对于不同的规格化的统一,则称之为一体化。这就是分布式数据库中的全局概念层到局部概念层的映射模式的描述。

4) 局部内层

局部内层是分布式数据库中关于物理数据库的描述,相当于集中式数据库的内层。其描述的内容和方法与之大致相同。

总之,分布式数据库 4 层结构及其模式定义之间的相互映射关系,说明分布式数据库是一组用网络连接的局部数据库的逻辑集合。4 层结构也体现了分布式数据库的特点。

(1) 全局数据库与局部数据库分离。全局数据库是虚拟的,全局 DBA 的视图由全局概念层定义,完全独立于各个场地的局部数据库。局部概念层和局部内层可视为局部数据库,它是全局数据库的内层。这样,不论是同构或异构型的分布式数据库,其全局数据库到局部数据库都是由映射模式解释的,所不同的是,同构型分布式数据库比异构型分布式数据库在映射模式上的复杂性较低。而对于全局用户来说,他们所关心的只是外层定义的视图,他们只须使用全局数据提供的语言去操纵分布式数据库,无须考虑各种模型转换、语言转换及场地分配等细节。全局数据库与局部数据库的分开描述,不仅体现了本章中关于分布式数据库的定义,同时也体现了它具有模式转换的透明性。

(2) 数据库的数据独立性。4 层结构中的全局外层是数据库的用户视图,可有多。全局概念层和局部概念层是分布式数据库的全局整体逻辑数据和局部整体逻辑数据的抽象。由于分布式数据库的分布特性决定了全局整体逻辑数据的抽象只有一个,而局部数据的逻辑抽象每个局部数据库都各有一个,当然也允许其中的某些逻辑抽象完全相同。这样,分布式数据库就具有了集中式数据库那样的数据独立性——逻辑数据独立性和物理数据独立性。

(3) 透明性。在全局概念层中,把数据的分片概念和数据的分配概念分别定义,从而可实现分布透明中的分片透明和分配透明的分离。所谓分片透明即用户完全只对全局关系操作,而不管如何在逻辑上把关系划分成片段关系。在全局概念层分片透明是最高程度的透明。分配透明是较低级的透明,要求用户在片段上而不是在全局关系上操作,不必考虑片段的存放位置。对用户而言,可在完全透明的情况下,对系统支持的由分片定义的所需片段进行操作,并由系统选择出适当的场地执行,从而实现了用户对用户的分布完全透明。这种分离对分布式数据库的设计是十分有利的,可在逻辑设计阶段考虑分片的划分要求,而在实现时再考虑数据分配问题。

(4) 数据冗余控制。冗余只在分配时才涉及,并且分布式系统提供了重复副本透明性。分布式系统还可提供比场地透明更低一级的透明性管理,即用户只要指定某个副本,

系统可对其他副本完成相应的操作,从而保证所有副本的完整性和可用性。

2. 数据分布

数据分布是分布式数据库系统中的基本问题,解决好这个问题对提高分布式数据库系统的效率和性能有积极的作用。所谓数据分布是指在分布式环境中通过合理地分布数据,提高数据操作的自然并行度,以达到执行效率最佳的目标。在构建分布式数据库系统的运行环境时,必须考虑数据如何分布在系统的各个场地上,或者说,必须考虑构成分布式数据库系统的各个组成部分各自如何使用数据的问题。所以,在分布式数据库系统中,同样存在着分布式数据库的设计问题。数据分布就是讨论这个问题,包括分布式数据库的逻辑划分和物理分配,以及用户对分布式数据库的划分或分配的感知程度(透明度)。

数据分布要研究的问题是如何在分布式数据库中放置数据,从而使得相关数据之间的相对位置最佳。如何分布数据,使它们的相对位置最佳,还需要考虑许多其他问题,例如:

(1) 如果一个场地上的存储空间不够,无法存放要访问的所有数据时该怎么办?

(2) 数据相对位置对查询优化有什么样的影响?

(3) 如何才能知道相对位置最佳的数据分布是否最大限度地利用了网络环境并行性?

对于数据分布问题,人们已经进行了大量的研究工作,其中主要的工作都是围绕着两方面进行的:“高效的数据划分问题”和“数据放置问题”。第一个方面是关于如何把数据划分开,使得使用率最高的数据能够被放置在性能最好的场地上。第二个方面是关于如何把已划分好的数据合理地放置在网络上以获得最好的执行效率,减少网络传输的数据量。数据的划分和放置是数据分布问题的两个方面,只解决其中任何一个都不能说已经解决了数据分布问题。数据分布是分布式数据库的特征,解决数据分布的策略一般有以下几种:

(1) 集中式:所有全局数据片段都安排一个在节点上。

这种分布策略把系统数据都存放在一个节点上,对数据的控制和管理也比较容易,数据的一致性和完整性能够得到保证。但是由于数据的检索和修改都必须通过这个节点,使得这个节点的负担过重,容易出现瓶颈。另外,系统对这个节点的依赖性也过多,一旦这个节点出现故障,将使整个系统崩溃,系统的可靠性相对较差。为了提高系统的可靠性,该节点的设备效能和可靠性都必须提高。

(2) 分割式:所有全局数据有且只有一份,它们被分割成若干个逻辑片段,每个逻辑片段被分别指派在特定的节点上,即对全局数据进行了划分。

这种分布策略充分利用各个站点上的存储设备,数据的存储量大。在存放数据的各个节点可自治地检索和修改数据,发挥系统的并发操作能力。同时,由于数据是分布在多

个节点上的,所以当某部分节点出现故障时,系统仍可运行,提高了系统的可靠性。对于全局查询和修改,所需的时间会比集中式长些,因为数据不在同一场地上,需要进行网络通信。

(3) 复制式:全局数据有多个副本,每个节点上都有一个完整的数据副本。

采用这种策略的系统可靠性较高,响应速度快。数据库的恢复也较容易,可从任意的场地得到数据的副本。但要保持各个节点上数据的同步修改,将要付出昂贵的代价。另外,整个系统的数据冗余很大,系统的数据容量也只是一个节点上数据库的容量。

(4) 混合式:全部数据被分为若干个数据子集,每个子集被放在不同的节点上,但任何一个节点都没有保存全部的数据。根据数据的重要性决定各个数据子集副本的数量。

这种分布策略兼顾了分割式和复制式的做法,也获得了二者的优点,具有一定的灵活性,能提高系统的效率,但同时也包括了二者的复杂性。

3. 数据分片

数据分片也称数据分割,是分布式数据库的特征之一。在一个分布式数据库中,全局数据库由各个局部数据库逻辑组合而成。相反,各个局部数据库则是全局数据库的某种逻辑分割而得。实际上这也是应用的需要,下面以关系模型为例来说明。一个关系描述了某些数据之间的逻辑相关性,但是,因为不同节点的用户对该关系中元组的需求可能是不同的。例如,某个关系中的元组与地区有关,西安的用户需要的是有关“西安”的那些元组,而广州的用户需要的是有关“广州”的那些元组。这就需要对这个关系进行分割,分割后得到的各部分元组称为该关系的逻辑片段,并被存放在相应的节点上。这样处理将各得其所,可以大大减少网络上的通信,从而提高系统的效率。

在分布式数据库中,数据存放的单位是数据的逻辑片段。对关系数据库来说,一个数据的逻辑片段是关系的一部分。数据分片有 3 种基本方法,它们是通过关系代数的基本运算来实现的:

(1) 水平分片:按特定条件把全局关系的所有元组分划成若干个互不相交的子集。每一子集为全局关系的一个逻辑片段,通过对全局关系施加适当的运算得到,并可通过对这些片段执行合并操作来恢复该全局关系。

(2) 垂直分片:把全局关系的属性分成若干子集,对全局关系进行投影运算即可得到这些子集。这要求全局关系的每一属性至少映射到一个垂直片段中,且每一个垂直片段都包含该全局关系的关键字。这样,通过对这些片段执行连接操作可以恢复该全局关系。

(3) 水平和垂直结合的分片:即以上两种方法的混合。可以先水平分片再垂直分片,或先垂直分片再水平分片。

不论采用哪一种方法进行数据分片,都要遵守下列规则。

(1) 完备性条件：必须把全局关系的所有数据映射到各个片段中，绝不允许有属于全局关系却不属于任何一个片段的数据存在。

(2) 可重构条件：必须保证能够由同一个全局关系的各个片段来重新构造该全局关系。对于水平分片可用并操作重构全局关系，对于垂直分片可用连接操作重构全局关系。

(3) 不相交条件：要求一个全局关系被分割后所得的各数据片段互不重叠或只包含关键字重叠。

4. 分布透明性

在分布式数据库中，数据独立性是十分重要的，其内容比集中式数据库更加复杂。除了数据的逻辑独立性与数据的物理独立性外，还有数据的分布独立性。所谓数据分布独立性是指用户或用户程序使用分布式数据库如同使用集中式数据库那样，不必关心全局数据的分布情况，即用户不必关心全局数据的逻辑分片情况、逻辑片段的场地理位置分配情况以及各场地上数据库的数据模型等。也就是说，全局数据的逻辑分片、片段的物理位置分配以及各场地数据库的数据模型等情况对用户和应用程序是透明的。所以，在分布式数据库中，分布独立性也称为分布透明性。下面介绍分布透明性的级别。

1) 分片透明性

分片透明性是分布透明性中的最高层。在 4 层分布式数据库模式结构中，分片透明性位于全局概念模式与分片模式之间。当分布式数据库具有分片透明性时，用户编写的应用程序只对全局关系进行操作，而不必考虑数据的逻辑分片。当分片模式改变时，只要改变全局概念模式到分片模式之间的映像，不会影响应用程序，从而实现了数据分片透明性。

2) 分配透明性

分配透明性是分布透明性的中间层，在 4 层的分布式数据库模式结构中，位于分片模式与分配模式之间。实际上，分配透明性包含两种情形：一种是各片段被复制的情况，即每一片段是否被复制、复制了几个副本；另一种是片段及其各副本的场地理位置分配情况。前者也称复制透明性或数据冗余透明性。当分布式数据库具有分配透明性时，用户编写的应用程序要了解全局数据的数据分片情况，但不必了解各逻辑片段的复制副本情况，也不必关心各片段及其副本的节点位置分配情况。当片段及其副本的存储节点改变时，只要改变从分片模式到分配模式之间的映像，不会影响用户程序，从而实现了数据片段的位置透明性。

3) 局部数据模型透明性

局部数据模型透明性也称局部映像透明性，它与各场地上数据库的数据模型无关，是分布透明性的最低层。在 4 层分布式数据库模式结构图中，处于分配模式与局部概念模式之间。当分布式数据库只具有局部数据模型透明性时，用户编写应用程序不但要了解

全局数据的逻辑分片情况,还要了解各逻辑片段的副本复制情况,以及各片段及其副本的站点位置分配情况,但不必了解各节点上数据库的数据模型。全局数据模型与每个节点上局部数据库的数据模型的转换是由分配模式与局部概念模式之间的映像实现的。当某个节点上数据库的数据模型改变时,只要改变分配模式到该节点局部概念模式之间的映像即可,应用程序不受影响,从而实现了局部数据模型透明性。显然,在同构分布式数据库系统中,各节点上的数据模型相同,且全局数据库的数据模型可能就是采用局部数据库的数据模型。此时,就大大减少了这种映像的复杂性。

如果一个分布式数据库系统提供分片透明性,它一定也提供分配透明性和局部数据模型透明性,所以也称为完全分布透明性,是分布透明性的最高级别。此时,对用户和用户程序而言,他们所面对的分布式数据库系统如同集中式数据库一样,不必考虑数据的分片细节,不必考虑各片段的副本情况,不必考虑片段及副本的分配细节,也无须考虑各节点上的数据库是什么数据模型等。

如果一个分布式数据库系统提供分配透明性,而没有提供分片透明性,它一定也提供局部数据模型透明性,所以也称为中级分布透明性。此时,对用户和应用程序而言,他们必须知道分布式数据库全局数据的逻辑分片情况,必须在程序中指出所需要访问的逻辑片段名。但不必关心逻辑片段是否被复制以及它们被分配在哪些节点上,也不必考虑节点的数据模型。

如果一个分布式数据库系统只提供局部数据模型透明性,而不提供分片透明性,也不提供分配透明性则称为低级分布透明性。此时,对用户和应用程序而言,他们不但必须知道分布式数据库全局数据的逻辑分片情况,还必须知道各片段是否有副本、有多少副本、各片段及其副本被分配在哪些节点上。即在程序中不但要指定需要访问的数据逻辑片段名,还要指定它们所在的节点名,但不必考虑节点上的数据模型。

如果一个分布式数据库系统连局部数据模型透明性也不提供,也即将异构数据模型转换也交给用户和用户程序自己处理,这种分布式数据库系统称为无分布透明性。

由此可见,一个分布式数据库系统可能提供的分布透明性的层次越高,用户编写应用程序越容易。因此,一个分布式数据库系统可能提供的分布透明性的程度,也是衡量分布式数据库管理系统是否完善的标准之一。

5. 分布式数据库管理系统

分布式数据库管理系统(distributed database management system, DDBMS)用以支持分布式数据库的建立和维护,Peebl 和 Manning 把分布式数据库管理系统分成综合型和联合型两大类。

(1) 综合型体系结构:是指在分布式数据库建立之前,还没有建立独立的集中式数据库管理系统。设计人员可根据用户的需求,设计出一个全新的完整的数据库管理系统。

(2) 联合型体系结构：是指每个节点的数据库管理系统已经存在，在此基础上建立的分布式数据库系统。同时，联合型体系结构又分为同构系统和异构系统。所谓同构系统是指每个节点的局部数据库管理系统支持同一种数据模式、命令语言及查询语言。而异构系统是指各个节点上的数据库管理系统有不同的数据模式、命令语言及查询语言。

1987 年，关系数据库的最早设计者提出了完全的分布式数据库管理系统应遵循的 12 条规则，这 12 条规则已被广泛接受，并作为分布式数据库系统的标准定义。它们是：

- (1) 场地自治性；
- (2) 非集中式管理；
- (3) 高可用性；
- (4) 位置独立性；
- (5) 数据分割独立性；
- (6) 数据复制独立性；
- (7) 分布式查询；
- (8) 分布式事务管理；
- (9) 硬件独立性；
- (10) 操作系统独立性；
- (11) 网络独立性；
- (12) 数据库管理系统独立性。

如果一个分布式数据库管理系统能够满足上面的 12 条准则，就可称这个分布式管理系统为完全的分布式管理系统。图 13-3 给出了一个分布式数据库管理系统的参考结构。

分布式数据库管理系统由 4 部分组成：

(1) 局部场地上的数据库管理系统(local database management, LDBMS)。其功能是建立和管理局部数据库，提供场地自治能力，执行局部应用及全局查询的子查询。

(2) 全局数据库管理系统(global database management, GDBMS)。主要功能是提供分布透明性，协调全局事务的执行，协调各局部数据库管理系统以完成全局应用，保证数据库的全局一致性，执行并发控制，实现更新同步，提供全局恢复功能等。

(3) 全局数据字典(global data directory, GDD)。包括模式、分片模式、分配模式及模式间映像的定义，存取权限的定义(保证全体用户的合法权限和数据库的安全性)，约束条件的定义，其功能与集中式数据库的数据字典类似。

(4) 通信管理(communication management, CM)。通信管理系统在分布数据库各场地之间传送消息和数据，完成通信功能。

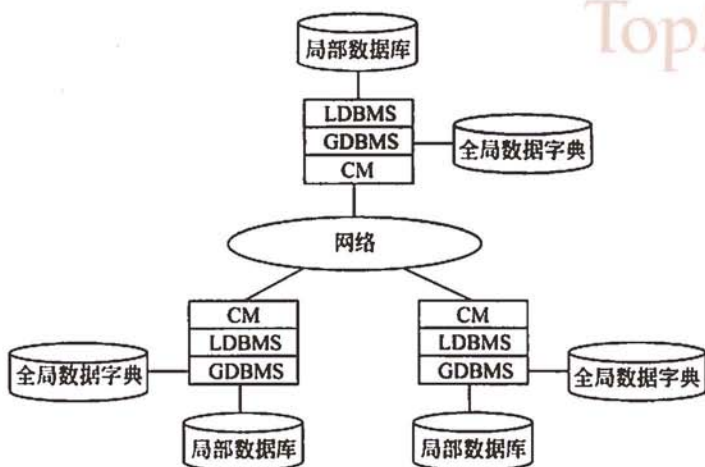


图 13-3 分布式数据库管理系统结构图

13.1.3 分布式查询处理和优化

在集中式关系型数据库中,数据的存取过程和数据的存储结构对用户和应用都是透明的,用户依据数据模型表达查询要求。借助于一些 SQL 语言来存取数据。这些数据存取及优化过程由数据库管理系统中的数据操作处理模块完成,它们决定了系统的效率。分布式数据库中查询处理的基础是集中式关系数据库系统中的概念及策略。下面从分布式查询的特点入手,讨论分布式查询处理,并假设分布式数据库管理系统提供完全透明性。

与集中式数据库环境中的查询相比,分布式数据库环境中的查询还要考虑以下两个方面:

- (1) 数据和信息均要通过通信线路进行传输,存在的延迟问题将减慢整个查询的执行过程。
- (2) 网络中多处理器的存在提供了并行数据处理和传输的机会,应充分利用以加快查询的速度。

在分布式数据库系统中,查询优化器的主要任务是控制和加快查询的执行与数据传输过程。在分布式查询处理技术中,查询优化的基本类型通常包括两类:针对查询执行代价的优化和针对查询响应时间的优化。执行代价是指查询所需要的系统资源,查询响应时间是指开始提交查询到获得第一个结果之间的时间。一般情况下,对一个组织机构而言,查询响应时间往往就代表着执行代价。例如对于一个商业组织机构而言,如果因为响应时间的延误而失掉了销售额或其他机会,就意味着商业损失。但是,出于教学和编程

方面的某些原因,还是需要将查询执行代价和响应时间加以区分。

针对查询执行代价进行优化的目标是,使查询执行所使用的系统资源的总和尽量少,从而降低系统开销。整个系统的开销可以从各单个系统资源的开销表达式中推出。针对查询响应时间优化的目标是尽量减少查询的响应时间,而不计较系统资源的耗费。可以形象地说,执行代价优化的目标是“最便宜”,而响应时间优化的目标是“最快”。

13.1.4 分布事务管理

1. 分布式事务

一个事务是访问数据库的一个逻辑工作单位,也可以说是一个操作序列,执行这个操作序列,使数据库从一种一致的状态切换到另一种一致的状态,以实现特定的业务功能。

分布式事务是传统事务的扩充。在分布式数据库系统中,任何一个应用的请求最终都将转化成对数据库的存取操作序列。所以从外部特征来看,分布式事务继承了传统事务的定义。但是,由于在分布式数据库系统中数据是分布的,一个事务的执行可能涉及到多个节点上的数据,使得分布式事务的执行方式与传统事务的执行方式不同。传统集中式事务只在一台计算机上执行,而分布式事务将在多个节点上的多台计算机上执行,即分布式事务的执行也是分布的。

在分布式数据库系统中,用分布式事务表明一个要求访问多个节点上数据库中数据的事务,但不关心存放数据的具体地点。分布式数据库管理系统的事务优化器用于把一个分布式事务转变为若干个与相应节点有关的操作序列,这些操作序列也称为“子事务”。所以,在分布式数据库系统中,可以认为一个分布式事务是由若干个不同站点上的子事务组成的。

2. 分布式事务的特性

分布式事务和集中式数据库中的事务一样具有下面的特性。

(1) 原子性:事务的操作要么全部执行,要么全不执行。当事务非正常终止时,其中间结果将被取消。事务的原子性保证数据库的状态总是从一个一致的状态变化到另一个一致的状态,而不会出现不一致的中间状态。

(2) 可串行性或一致性:并发执行的几个事务,其操作的结果应与以某种顺序串行执行这几个事务所得出的结果相同,因此称为可串行性。这种可串行化的并行调度是由数据库系统的并发控制机制实现的,以保证并发事务执行时数据库状态的一致。所以这种性质也称为事务的一致性。

(3) 隔离性:一个没执行完的事务不能在其提交之前把自己的中间结果提供给其他的事务使用。因为未提交事务的结果不是最终结果,有可能在以后的执行中被迫取消。如果其他事务用到了它的中间结果,那么该事务也要夭折。

(4) 持久性: 当一个事务正常结束后, 即提交后, 其操作的结果将永久化, 提交后发生的故障不会影响提交结果。即使发生了故障, 系统应保证能够把事务的操作结果恢复过来。

人们常把事务的原子性 (Atomicity)、可串行性 (Serializability) 或一致性 (Consistency)、隔离性 (Isolation) 以及持久性 (Durability) 简称为事务的四性, 即 ACID 或 ASID 或 ACID。

由于分布式数据库系统的分布特性, 分布式事务的四性更带有分布执行时的特性。例如, 在分布式数据库系统中, 为了保证事务的原子, 组成这个分布式事务的各个子事务, 要么全部都提交 (成功结束), 要么全都撤销 (不成功结束), 这就需要对各子事务进行协调和控制。此外, 在分布式事务中, 除了需要考虑访问数据互斥的存取操作序列外, 还必须考虑大量的数据传送、通信原语和报文控制等, 这些都是分布式事务所特有的性质。因此分布式事务与集中式数据库中的事务相比, 在下面几个特性有所区别。

(1) 执行特性: 由于分布式事务执行时被分解成多个子事务, 而各子事务间的操作需要进行协调, 因此每一个分布式事务必须创建一个控制进程 (亦称协调进程), 以协调各子事务的操作, 协调数据及控制报文的收发, 决定事务的提交与夭折。而集中式事务的执行由并行调度算法调度, 不必产生一个控制进程, 也不必分解为子事务。

(2) 操作特性: 在分布式事务中, 除了应用对数据的存取操作序列之外, 还必须加入大量的通信原语, 负责协调进程和代理进程 (负责完成子事务) 之间的数据传送, 以及代理进程之间的数据传送。此外, 为了协调子事务的执行, 还要加入大量的控制原语。因此, 分布式事务比集中式事务的组成要复杂, 而且执行的方式也要复杂得多。

(3) 控制报文: 分布式数据库系统中, 除了数据报文外, 更增加了控制报文。因为除了要对数据进行存取操作外, 还要对各子事务的操作进行协调, 这样就有了大量的控制报文需要在网上传输。

3. 分布式数据库故障

在集中式数据库系统中, 故障分为事务故障、系统故障和介质故障。

在分布式数据库系统中, 除了上述故障外还有因网络引起的故障。一般把网络上各节点可能出现的故障称为节点故障, 其中包括集中式系统中可能发生的故障, 而把节点之间通信出现的故障称为通信故障。

通信故障可分为报文故障和网络分割故障。而报文故障又可分为报文错、报文失序、报文丢失和长时间的延迟。对于报文错和报文失序, 现今的网络都可检测和处理。所以通信故障主要是报文丢失、报文延迟和网络分割。下面我们对每一种故障做一个解释。

(1) 介质故障: 存放数据的介质发生的故障, 如磁带、磁盘的损坏等。

(2) 系统故障: CPU 错、死循环、缓冲区满及系统崩溃等。

(3) 事务故障：计算溢出、完整性被破坏、操作员干预及输入输出错等。

(4) 网络分割故障：系统中的一部分节点和另外一部分节点完全失去了联系，两组节点无法通信。

(5) 报文故障：收到的报文格式或数据错误、报文先后次序不正确、丢失了部分报文和长时间收不到报文。

综上所述，故障的分类如图 13-4 所示。

当在规定时间内未接到应答时，就要进行如下的分析：

(1) 系统发生故障，性能不好，还是网络流量过大？

(2) 如果系统发生故障，是节点故障，报文故障，还是网络分割？

(3) 如果是报文故障，是报文丢失还是应答丢失？

等等。

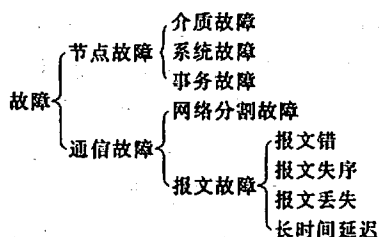


图 13-4 故障的分类图

处理网络分割故障要比处理节点故障和报文故障因难得多，但其发生频率也低于节点故障和报文故障。按照故障处理难度的升序排列，则为：

(1) 仅发生节点故障；

(2) 节点故障与报文故障同时存在；

(3) 节点故障、报文故障和网络分割故障同时存在。

4. 分布式数据库的恢复原则

故障的发生会影响数据库中数据的正确性，甚至破坏数据库，从而影响数据库系统的可靠性和可用性。因此，数据库管理系统都对故障恢复机制很下功夫，认真地做了研究和开发。研究数据库系统中故障的恢复，主要是指如何恢复因故障而破坏的数据库，使数据库恢复到正确状态。由前面的分析可知，在分布式数据库系统中，故障的类型很多。当发生事务故障时，保证事务原子性的措施称为事务故障恢复，简称为事务恢复。事务本身的故障和系统的故障是造成数据库完整性和一致性破坏的主要原因。事务恢复主要靠日志来实现。恢复应遵循的原则如下。

1) 孤立和逐步退出事务的原则

对于不影响其他事务的可排除性局部故障，例如事务操作的删除、超时、违反完整性规则、资源、限制以及死锁等，应令某个事务孤立地和逐步地退出，将其做过的修改复原，即做 UNDO。

2) 成功结束事务原则

成功结束事务所做过的修改应超越各种故障。当故障发生时，应该重做 (REDO) 事务的所有操作。

3) 夭折事务的原则

若发生了非局部性的不可排除的故障,例如系统崩溃,则撤销全部事务,恢复到初态。这两种做法,一种是利用数据库的备份实现,另一种是按反向顺序操作,对启动以来所做过的一切修改进行复原。

从集中式事务恢复可以了解事务恢复的一般过程。对分布式事务来说,由于处于网络环境,其恢复处理远比集中式事务恢复要复杂得多。在分布式事务恢复中,本地事务的恢复和集中式事务的恢复相同,由本地事务管理器(Local Transact Management,缩写 LTM)具体执行。而整个分布式事务的恢复由分布式事务管理器(Distribute Transact Management,缩写 DTM)与 LTM 协同完成。

5. 两阶段提交协议

两阶段提交协议(Two Phase Commitment Protocol, 2PC)既简单又精巧,它把本地原子性提交行为的效果扩展到分布式事务,保证了分布式事务提交的原子性,并在不损坏日志的情况下,实现快速故障恢复,提高分布式数据库系统的可靠性。

在两阶段提交协议中,把分布式事务的某一个代理指定为协调者(Coordinator),所有其他代理称为参与者(Participant)。这里的代理是指完成各个子事务的进程。只有协调者才拥有提交或撤销事务的决定权,而其他参与者则各自负责在其本地数据库中执行写操作,并向协调者提出撤销或提交事务的意向。一般一个节点惟一对应一个子事务,如果某一参与者与协调者在同一节点,虽然它们不需要使用网络来通信,但仍从逻辑上认为它与协调者不在同一节点。图 13-5 描述了协调者和参与者的关系。

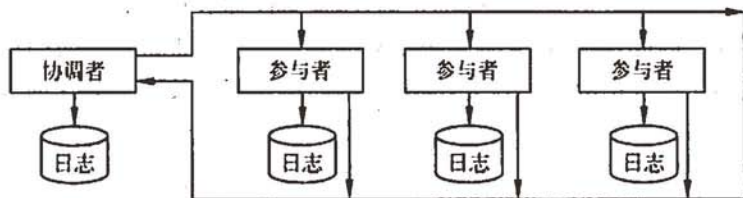


图 13-5 协调者和参与者的关系图

2PC 之所以能保证分布式事务提交的原子性,这是因为在分布式事务的结果生效以前,所有参与执行分布式事务的节点都同意提交而做到这一点的。有很多理由可以说明这种同步的必要性,如果某个事务正在读一项由另一个还未提交的事务更新的数据项的值时,相应的参与者就不会同意马上提交该事务。另一种参与者不同意提交的可能原因是发生了死锁,这要求某一个参与者撤销事务。注意,参与者不需要其他任何进程来通知就可以撤销一个事务,这种能力相当重要,我们称之为单方面撤销。

2PC 把事务的提交过程分为两个阶段:

第一阶段是表决阶段,目的是形成一个共同的决定。开始时,协调者在它的日志中写入一条开始提交的记录,再给所有参与者发送“准备提交”消息,并进入等待状态。当参与者收到“准备提交”消息后,它检查是否能提交本地事务。如果能提交,参与者在日志中写入一条就绪记录,并给协调者发送“建议提交”消息,然后进入就绪状态。否则,参与者写入撤销记录,并给协调者发送“建议撤销”消息。如果某个节点做出“建议撤销”提议,由于撤销决定具有否决权(即单方面撤销),发出“建议撤销”的站点就可以直接忽略这个事务。协调者收到所有参与者的回答后,就可以做出是否提交事务的决定。只要有一个参与者建议撤销,协调者就必须从整体上撤销整个分布式事务。因此写入一条撤销记录,并给所有参与者发送“全局撤销”消息,然后进入撤销状态。否则,写入提交记录,给所有的参与者发送“全局提交”消息,然后进入提交状态。

第二阶段是执行阶段,目的是实现这个协调者的决定。根据协调者的指令,参与者或者提交事务,或者撤销事务,并给协调者发送确认消息。此时,协调者在日志中写入一条事务结束记录并终止事务。图 13-6 描述了两阶段提交协议的参与者和协调者之间的交互过程。

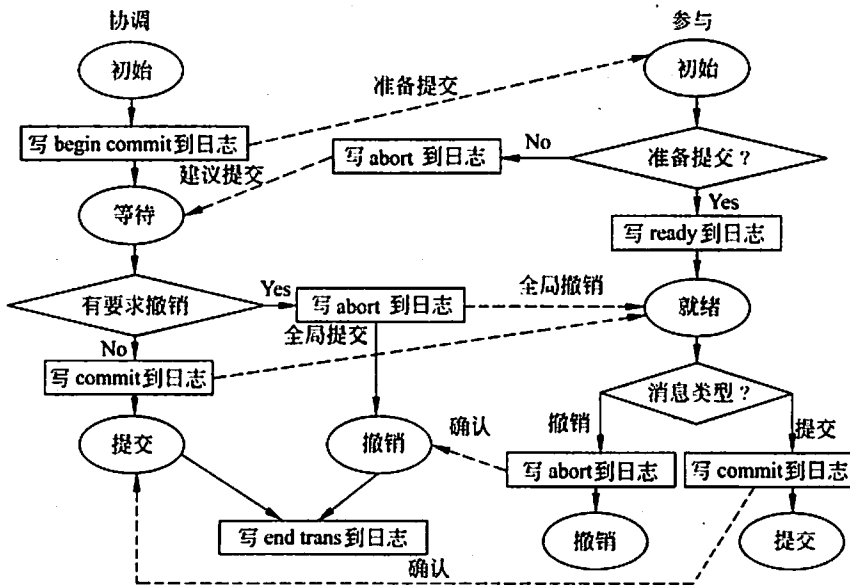


图 13-6 两阶段提交协议活动图

请注意协调者做出事务全局终止决定的方式,该决定受如下两条规则的支配,这两条规则称为全局提交规则:

- (1) 只要有一个参与者撤销事务,协调者就必须做出全局撤销决定;

(2) 只有所有参与者都同意提交事务,协调者才能做出全局提交决定。

从图 13-6 中可以看出以下一些关于两阶段提交协议的重要之处:

(1) 两阶段提交协议允许参与者单方面撤销事务;

(2) 一旦参与者确定了提交或撤销提议,就不能再更改它的提议;

(3) 当参与者处于就绪状态时,根据协调者发出的消息的种类,参与者可以转换为提交状态或撤销状态;

(4) 协调者依据全局提交规则做出全局终止决定;

(5) 注意,协调者和参与者可能进入某些相互等待对方发送消息的状态。为了确保它们能够从这些状态中退出并终止,要使用定时器。每个代理进程进入一个状态时都要设置超时器。如果所期待的消息在定时器超时之前没有到来,定时器向代理进程报警,进程根据超时协议执行相应的动作。

6. 两阶段提交协议对故障的恢复

1) 场地故障

- 当一参与者在写入“建议提交”前发生故障时,该参与者无法向协调者发回答信息。因此,当协调者等至超时后,将决定终止事务。故障恢复后,该参与者无须收集其他场地的信息即可终止事务。
- 参与者进程在写入“建议提交”后发生故障,这时其他的参与者可以正常地结束该事务——“提交”或“撤销”。因为协调者可以根据收到该参与者的应答决定“提交”或“撤销”。因此,故障恢复后,该参与者要访问协调者或其他参与者,以了解协调者对事务做出的决定,然后执行相应的操作——“提交”或“撤销”。这里我们假设在日志中写入“建议提交”记录和发送“建议提交”信息给协调者这两个动作具有原子性,要么都执行,要么都不执行。
- 协调者在日志中写入“准备提交”记录后,写入“全局提交”或“全局撤销”前发生故障时,已发出“建议提交”信息的参与者将等待协调者恢复。协调者的重新启动过程从头恢复提交协议,从“准备提交”记录中读出参与者的标识符,重发“准备提交”报文给参与者,重新执行提交过程。
- 协调者在写入“全局提交”或“全局撤销”记录以后,在写入“事务结束”记录以前发生故障。在这种情况下,协调者恢复时必须给所有的参与者重发其决定,未收到信息的参与者不得不等待协调者的恢复。

2) 报文丢失故障

- 至少有一个参与者的回答报文(“建议提交”或“建议撤销”)丢失了。在这种情况下,协调者将因等待回答而超时,整个事务被撤销。这种情况只有协调者能够发现,但它无法知道是场地故障还是通信故障。而参与者能够正确执行,因此不会

启动恢复过程。

- 丢失“准备提交”报文。由于至少有一个参与者收不到“准备提交”命令，因此参与者将处于等待状态，而协调者也在等待参与者的回答，所以协调者会因为等待超时而撤销事务。这种情况和上述一样。
- 丢失“全局提交”或“全局撤销”报文。在这种情况下，参与者将处于等待协调者命令的状态。当参与者未收到命令时，会因等待而超时，这时向协调者请求重发该命令的信息。
- 丢失了“确认”报文。当协调者未收到全部参与者的“确认”报文时，协调者会因等待而超时。这时协调者重发命令报文给参与者，参与者必须以报文给予“确认”回答，即使此时相应的子事务已不再活动也要重发。

3) 网络分割故障

假设在实现网络分割时，整个网被分为两个组，包含协调者的组称为协调者组，其他的则组成参与者组。对于协调者来说，这种情况相当于参与者组中的多个参与者同时发生故障，这时协调者可以做出决定，然后把命令发给协调者组中的参与者。因此这些场地上的子事务可以结束。而对于失去联系的参与者，它们则认为协调者出现了故障，根据漏收的回答信息，进行相应的故障处理。

7. 三阶段提交协议

所谓事务的阻塞是指一个场地的子事务本来是可以执行并正常结束的，然而由于分布式数据库的故障，它必须等待故障恢复以后得到需要的信息后才可以做出决定，而故障情况是无法预料的，该子事务又占有的一些系统资源不能释放，无法继续执行。这时称之为事务进入阻塞状态。

事务出现阻塞的原因可能很多，例如，即当参与者等待协调者的回答时，可能因为网络故障或协调者故障使之收不到回答信息而出现等待超时。这时事务进入阻塞状态，并重发“建议提交”信息，要求协调者给予回答。直到网络故障或协调者恢复并给予回答，参与者才能做出决定是否继续执行（提交或撤销）。若一直收不到回答，则事务会一直处于阻塞状态而挂在相应的场地上。因此，阻塞降低了事务的可用性。

如何使一提交协议成为非阻塞的提交协议呢？在 2PC 协议中，参与者的提交是在它知道了其他所有的参与者均发出了“建议提交”的报文以后才进行的。若在 2PC 中增加一个阶段，使得参与者的提交不仅要等到它知道所有的参与者均发出了“建议提交”的报文，而且还知道所有参与者的状态（如它们处于故障状态，还是已经恢复）以后才执行。这时 2PC 即变成 3PC 协议，即三阶段提交协议。在 3PC 协议中，报文有三次接收和发送，协调者第二次向参与者发出的报文不是“全局提交”报文，而是提交前的“全局预提交”报文，告诉所有的参与者均可以进入准备提交状态，而参与者的回答也不是提交子事务，而

是发出“准备就绪”报文。在第三阶段中,当协调者收到全部的“准备就绪”回答时才向所有的参与者发“全局提交”报文。此时,所有的参与者均知道其他的参与者已经进入“准备提交”状态。到达这一步时,每个参与者均可以自己做出决定——撤销或提交,而不必因等待协调者的回答而进入阻塞状态。因为即使此时发生故障,系统的恢复机制迟早会恢复到故障前一刻的状态,即各参与者的子事务总会提交。因此,参与者可以自行决定先执行下去而不是处于等待状态,从而减少了阻塞。3PC 协议的上述过程为:

第一阶段,协调者向所有的参与者发“准备提交”报文,每个参与者根据自己的情况进行投票。只有所有的参与者均回答“建议提交”才进入第二阶段。

第二阶段,协调者向所有的参与者发“全局预提交”报文,参与者收到该报文后若已经准备好提交,则回答“准备就绪”报文。否则进行撤销处理。

第三阶段,协调者收到所有参与者“准备就绪”的回答后,就向所有的参与者发“全局提交”报文。此时,每个参与者都知道其他参与者赞成提交,因此可以在收到“全局提交”报文后进行提交。

3PC 可以避免阻塞是基于一定的故障模型的。如果发生了网路分割故障,采用 3PC 协议同样存在问题。没有一种协议能够解决所有的故障,3PC 协议仅仅降低了阻塞发生的可能性,但不是完全的非阻塞协议。

8. 三阶段提交协议对故障的恢复

由于系统的故障,报文可能没有收到。与 2PC 一样,3PC 也采用超时方法处理。

第一种情况是协调者没能及时发出“准备提交”报文,导致参与者等待超时。这时参与者决定撤销。

第二种情况是协调者等待参与者投票结果超时。这时协调者决定撤销。

第三种情况是参与者处于“赞成”提交状态,等待“全局预提交”命令时出现超时。这时进入恢复处理过程。

第四种情况是参与者处于“准备就绪”状态,在等待协调者的“全局提交”命令出现超时。也进入恢复处理,这时只要有至少一个参与者处于活动状态,子事务就不会阻塞。因为恢复后的参与者可以从活动的子事务得到有关提交处理的信息,从而得知协调者的决定。依据协调者的决定确定自己应做的处理。

在进入恢复前,事务的下述两种状态是不相容的。

(1) 一个参与者在其他任何一个活动的参与者处于赞成提交状态时,不可能进入“提交”状态。

(2) 一个参与者在另一个参与者进入提交状态或任何一个参与者都进入准备就绪状态时不能进入撤销状态。

因此,恢复时参与者可以根据活动事务的状态决定相应的处理。在 3PC 协议中,恢

复制机制惟一可以做的是就近访问一个活动的参与者,确切地说就是访问在它进入恢复处理前最近的活动参与者。如果所有的参与者处于“赞成”提交或“撤销”状态,则肯定没有一个参与者已提交,因此可以通知全部参与者“撤销”。如果已经有一个参与者“准备就绪”或“提交”,则肯定没有一个参与者被“撤销”,所以可以通知全部参与者提交。在通知“全局提交”前应先使仍处于“赞成”提交的参与者进入“准备提交”状态,然后再进入“提交”状态。因为在通知提交前,任何一个参与者均可能再发生故障,所以应避免其中一个参与者处于“赞成”提交状态而其他的处于“提交”状态。

13.1.5 分布式数据库系统的应用

分布式数据库系统的必要性是显而易见的。由分布式数据库系统的特点可知,在一个各场地有数据库的计算机网络系统中;若有分布式数据库管理系统,则可使系统中的数据库实现共享,而且利用率高,存取快,可靠性高,用户使用这些数据库如同使用本地数据库一样,极为方便。反之,若没有分布式数据库管理系统,只利用现有的一般网络操作系统来存取网络中的数据,用户就会感到相当麻烦和困难,而且这类系统无法在有一定数据冗余(这是获得可靠性所需要的)的情况下保证数据的一致性,故障点的自动切除、故障后的恢复以及并发控制等问题就更没有保障了。所以,研制各种性能优良的分布式数据库系统,无论是同构还是异构型,都是极为必要的。

分布式数据库系统将广泛用于各企事业单位的人事、财务和库存等管理信息系统,百货公司销售点的经营信息处理系统,电子银行等的在线处理系统,国家政府部门的经济信息系统,大规模数据资源如人口普查、气象预报、环境污染和地震监测等数据库系统,军事指挥控制系统,具有多数据处理中心的企业生产系统,医院的病历药品管理、辅助诊断和病人监护系统,集散型的工业控制和管理系统,自动化综合制造系统,以及各种办公自动化系统等。

13.2 Web 与数据库

随着网络的高速发展和网络服务的日趋完善,网络上的信息量呈几何级数增加。为了有效地组织、存储、管理和使用网上的信息,数据库技术被普遍地应用于网络领域。20世纪90年代的数据库技术已经十分成熟,可以支持大容量数据的存储和检索,而且具有较高的稳定性和可靠性。如今,人们在Internet上建立了数以万计的网站,尤其是大中型网站的后台都有数据库系统的支持。数据库系统可以将网站的各种数据很好地组织起来,并自动生成Web页面。根据浏览者需求的不同,显示不同的页面内容,同时还可以实现一些逻辑操作。现在,没有数据库的支持,大中型网站将无法正常运行。

13.2.1 Web 概述

1. Web 数据库

Web 的产生是与互联网的发展密切相关的。互联网是由全球众多的计算机局域网互相连接组成的一个超大规模的网络系统。在这个系统中运行着多种应用系统,如上网使用的网页浏览系统——WWW,上传与下载用的文件传输系统——FTP,以及收发电子邮件所使用的电子邮件系统——E-Mail 等。互联网中运行的每一种应用系统都是由互联网中相应的服务器系统和客户机系统构成的,也就是说,从物理连接来看互联网是由众多的计算机组成的,而从逻辑上看互联网是由多个功能子网组成的。

在浏览网页时,服务器上的 WWW 服务允许用鼠标点击“超级链接”,每点击一项,WWW 程序就会执行所要求的任务,一直到需要得到满足。

在这一过程中,涉及两个不同的程序。一个程序安装在客户机上,处理鼠标点击事件,发出 http 请求。接到响应后,立即显示链接的网页内容。这个程序叫做 WWW 客户机程序,如上网使用的浏览器(IE 或 Netscape)。另一个程序在服务器上,如 IIS 或“阿帕奇”(Apache)Web 服务器软件。它对 WWW 客户机的请求进行处理并返回处理结果。也就是在接到 http 请求后,发出响应。

数据库是指按照一定的结构和规则组织起来的相关数据的集合,是存放数据的“仓库”。据此可以将网络数据库定义为以后台数据库为基础,加上一定的前台程序,通过浏览器完成数据存储和查询等操作的系统。数据库技术是计算机处理与存储数据最有效、最成功的技术,而计算机网络的特点是资源共享。因此,数据处理与资源共享这两种技术的结合即成为今天广泛应用的 Web 数据库(也叫网络数据库)。

用户利用浏览器作为输入接口,输入所需要的数据,浏览器将这些数据传送给网站,而网站再对这些数据进行处理,例如,将数据存入后台数据库,或者对后台数据库进行查询操作等,最后网站再将处理结果传回浏览器,通过浏览器将结果告知用户。网站上的后台数据库就是 Web 数据库。

2. WWW 网络环境下的 Web 数据库

由于 Web 的易用性和实用性,它很快占据了主导地位,并且已经成为使用最为广泛、最有前途和最有魅力的信息传播技术。不过,Web 服务只是提供了 Internet 上信息交互的平台。随着 Internet 技术和 Web 技术的蓬勃发展,人们已不满足于只在 Web 浏览器上获取静态的信息,而是需要通过它发表意见、查询数据,甚至进行网上购物,这就迫切需要提供全方位的 Internet 服务。

Web 与数据库的结合,将人、企业、社会与 Internet 融为一体。Web 技术发展到今天,人们已经可以把数据库技术引入到 Web 系统中。数据库技术发展比较成熟,特别适

用于对大量的数据进行组织和管理。Web 技术具有较佳的信息发布途径,这两种技术的天然互补性决定了相互融合是其发展的必然趋势。将 Web 技术与数据库技术融合在一起,使数据库系统成为 Web 的重要有机组成部分,不仅可以把二者的所有优点集中在一起,而且能够充分利用大量已有的数据库信息资源,使用户在 Web 浏览器上方便地检索和浏览数据库的内容,这对许多软件开发者来说具有极大的吸引力。因此,将 Web 技术与数据库相结合,开发动态的 Web 数据库应用已成为当今 Web 技术研究的热点。

关系数据库最初设计为基于主机/终端方式的大型机上的应用,其应用范围较为有限。随着客户机/服务器方式的流行和应用向客户机方向的分解,关系数据库又经历了客户机/服务器时代,并获得了极大的发展。

随着 Internet 应用的普及,由于 Internet 上信息资源的复杂性和不规范性,关系数据库初期在开发各种网上应用时显得力不从心,表现在无法管理网上的各种复杂的文档型和多媒体型数据资源。后来关系数据库对于这些需求做出了一些适应性的调整,如增加数据库的面向对象成分以增强处理多种复杂数据类型的能力,增加各种中间件(主要包括 CGI、ISAPI、ODBC、JDBC 和 ASP 等技术)以扩展基于 Internet 应用的能力,通过应用服务器解释执行各种 HTML 中嵌入脚本来解决 Internet 应用中数据库数据的显示、维护、输出以及到 HTML 格式的转换等。

此时的基于 Internet 应用的关系数据库的典型模式表现为一种三层或四层的多层结构。在这种多层结构体系下,关系数据库解决了数据库的 Internet 应用的方法问题,使得基于关系数据库能够开发各种网上数据库数据的发布、检索、维护及管理一般性应用。

但是,关系数据库从设计之初并没有也不可能考虑到以 http 为基础,以 HTML 为文件格式的互联网的需求,只是在互联网出现后才做出相应的调整。

同时,关系数据库的基于中间件的解决方案又给 Internet 应用带来了新的网络瓶颈,应用服务器端由于与数据库频繁交互,其本身的效率和数据库检索的效率导致 Internet 应用存在应用服务器端的阻塞。

虽然关系型数据库具有完备的理论基础、简洁的数据模型、透明的查询语言和方便的操作方法等优点,但由于它本身并没有针对网络的特点和要求进行设计,并不适用于网络环境,因此我们应该研究开发新的数据库技术,

从一开始就考虑 Web 信息和结构的特点,使数据库真正能与 Web 融合为一体,充分利用二者的特点,建立合理的 Web 数据库。

那么,为什么使用网络数据库呢?简言之,因为人们在网络数据库中可以找到他们需要的东西,包括期刊查询,检查银行账户、股票价格、利率、实现电子商务等等。这些功能是用 HTML 编写的网页无法做到的,因为用 HTML 无法完成交互功能。要做到这些,必须使用网络数据库技术。

Web 数据库可以实现方便廉价的资源共享。数据信息是资源的主体,因而网络数据库技术自然而然成为互联网的核心技术。

当前比较流行的 Web 数据库主要有 SQL Server、MySQL 和 Oracle。这三种数据库适应性强,性能优异,容易使用,在国内得到了广泛的应用。

13.2.2 Web 服务器脚本程序与服务器的接口

Web 页面与数据库的连接是 Web 数据库的基本要求。目前,基于 Web 数据库的连接方案主要有两种类型:服务器端和客户端方案。服务器端方案的实现技术有 CGI、SAPI、ASP、PHP 和 JSP 等;客户端方案的实现技术有 JDBC (Java Database Connectivity)和 DHTML(Dynamic HTML)等。其中 ASP 是微软开发的脚本语言技术,嵌入在 IIS 中。因此,ASP 也就顺理成章地成为大部分 Windows 用户首选的脚本语言。

通常,Web 数据库的环境由硬件元素和软件元素组成。硬件元素包括 Web 服务器、客户机、数据库服务器和网络。软件元素包括客户端和服务端两个方面。客户端必须有能够解释执行 HTML 代码的浏览器(如 IE 和 Netscape 等)。在 Web 服务器中,必须具有能自动生成 HTML 代码程序的功能,如 ASP 和 CGI 等,具有能自动完成数据操作指令的数据库系统,如 Access 和 SQL Server 等。

Web 服务器使用 HTTP 协议对客户机的请求给予回答。每个 Web 服务器在 Internet 上都有一个惟一的地址,这个地址可以是一个域名,也可以是 4 个以点分隔的 0 到 255 之间的数字。例如 www.yahoo.com 或 202.106.168.67。如果客户机提出一个合法的请求,那么 Web 服务器就会把请求的内容传送给客户机。Web 服务器不仅能够传送各种文件,还可以传送某个程序的输出结果,这给 Web 和数据库的结合创造了条件。Web 服务器的种类很多,比较著名的有 IIS 和 Apache 等。

图 13-7 给出了典型的 Web 和数据库的运行模式。

在脚本程序中一般都需要采用相应的接口连接数据库。连接数据库的常用方法有 ODBC、DAO、RDO 及 ADO 等。

(1) ODBC ODBC(Open Database Connectivity,开放式数据库连接)是微软开发的一套统一的程序接口。通过这个接口可以存取不同厂商生产的数据库。经过多年的改进,它已成为存取数据库服务器的标准。事实上,ODBC 技术已成为后来的 DAO、RDO 及 ADO 等数据库访问技术的基础。

(2) DAO DAO(Data Access Objects,数据访问对象)是微软公司开发的一套主要的应用程序及开发工具,利用它可以访问数据库的标准对象,如 Access、VB、Excel 和 Word 等。

(3) RDO RDO(Remote Data Objects,远程数据对象)是微软公司为增强 DAO 的功

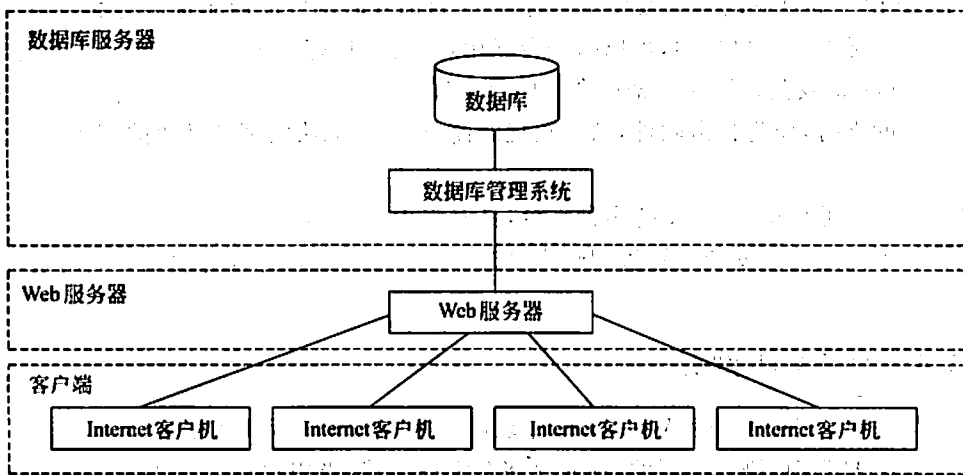


图 13-7 Web 和数据库的运行模式图

能而推出的新产品。该产品强化了 SQL Server 的访问功能,提高了 SQL Serve 的执行效率。

(4) ADO ADO(ActivteX Data Objects, ActivteX 数据对象)是微软在 Internet 领域采取的新举措。它本身并不是一项新技术,从对象结构的角度来看,它比 DAO 提供的对象更少。从存取 SQL 服务器的角度来看,它提供的功能也不如 RDO。但它汲取了 DAO 和 RDO 最精华的部分,成为一个更适合于 Internet 的小而精的对象群。因此,ADO 实际上是脚本程序连接数据库的一种最佳选择。

13.2.3 应用开发平台

由于 Web 应用开发的独特性,应用开发平台成为众多厂商关注的焦点。目前市场上存在很多 Web 应用标准和集成开发环境,其中最流行的主要是 ASP、PHP 和 JSP 3 种。

1) ASP

ASP(Activex Server Pages)是由微软创建的 Web 应用开发标准。ASP 服务器已经包含在 IIS 服务器中,ASP 服务器将 Web 请求转入解释器中,在解释器中对所有 ASP 中的脚本进行分析,然后执行。同时它还可以创建 COM 对象以完成更多的功能。ASP 中的脚本是 Vbscript,其优点是安装配置方便,开发简单易学,开发工具功能强大。不足之处是 ASP 使用了组件,因而有可能导致大量的安全问题,无法跨平台应用,只适用于 Windows NT/2000 环境。

2) PHP

PHP 由于其良好的性能及免费的特点,是目前互联网中应用非常流行的一种应用开发平台。其优点是简单易学,可以跨平台互用,是具有良好数据库交换能力的开发语言,能够与 Apache 及其扩展库紧密结合,具有良好的安全性。不足之处是安装配置复杂,缺少企业级的支持,作为自由软件,缺乏正规的商业支持,无法实现商品化的商业开发。

3) JSP

其优点是可移植性好,支持多种平台,具有强大的可伸缩性,具有多样化与强大的工具支持。不足之处是安装配置管理较为复杂,运行速度较慢,建议开发大型应用系统时采用 JSP。

13.2.4 动态 Web 网页

在静态 Web 网页阶段,Web 技术主要用于简单的静态 Web 页面浏览,静态 Web 页面都是采用 HTML 语言编写的,预先存储在服务器端,当用户请求该 Web 页面时,通过网络将该页面传送到客户端。用户使用客户端的浏览器,通过站点主页进入 Internet 上的各个 Web 站点进行访问。此时,由于受到低版本 HTML 和老式浏览器的制约,Web 页面只能包含单纯的文本内容,因而只能在浏览器中显示呆板的文字信息。随着 Internet 上信息的日益丰富,原有的文本浏览已不能满足于广大用户的需求。

随后,由于 HTML 标识不断扩充,浏览器不断改进才使 Web 页面逐渐支持各种媒体文件。所以,这个阶段的 Web 服务器基本上只是一个 HTTP 服务器,它负责接收客户端浏览器的访问请求,建立连接,响应用户的要求,查找所需要的静态 Web 页面,返回给浏览器进行显示。图 13-8 显示了浏览 Web 页面的体系结构。

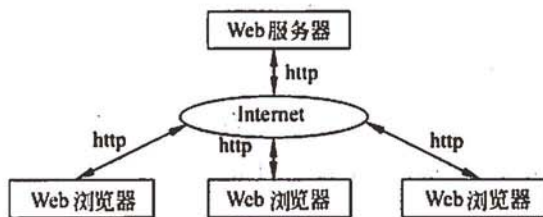


图 13-8 浏览 Web 页面的体系结构图

由于静态页面无法使用户与之交互,所以用户只能被动地使用 Web 服务器所提供的静态页面。另一方面,对于网络管理员来说,维护静态 Web 页面也是一件很麻烦的事,需要不断地创建新的 Web 页面并且不断删除与修改已有的 Web 页面,因而越来越不能满足人们的进一步要求。随着高版本 HTML 的出现和浏览器对表单的支持,Web 技术进入了动态交互页面阶段。

在动态交互页面阶段初期,只是对用户交互信息进行简单的处理,按照用户的要求对主页进行有限的控制。随着动态信息内容的不断加大,以及数据库信息资源发布的需求,越来越多的技术人员认识到 Web 与数据库连接的重要性。网络数据库技术就是此阶段最重要的技术核心。图 13-9 显示了通过 Web 浏览器访问数据库的各种方式。

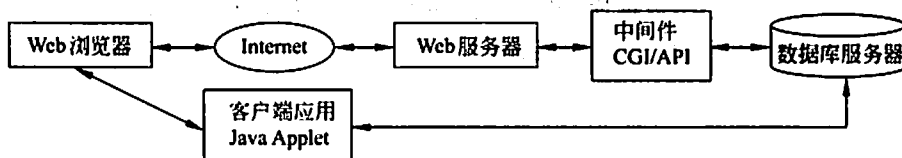


图 13-9 Web 浏览器访问数据库方式图

13.2.5 CGI 的应用

公共网关接口(Common Gateway Interface,缩写 CGI)是最早出现的动态网页发布技术,由于其开发较早,技术成熟,目前仍是动态网页开发的主力之一。Common 表示 CGI 可以确保使用多种程序语言和多种不同的系统交互。Gateway 表示 CGI 的力量并不限于它本身所做的事,而在于它提供了连接其他系统的潜力,例如数据库和图形生成工具等。Interface 表示 CGI 对如何更好地利用其特性提供了明确的定义。换句话说,可以设计程序来适当地利用这个接口。CGI 是 Web 服务器调用外部程序的接口。通过 CGI, Web 服务器能完成一些本身所不能完成的工作。早期很多著名的服务器都以自己独特的方式,支持服务器端的可执行程序,用来帮助完成客户机的请求。为某个服务器写的程序若要用于其他服务器时,就必须做较大的修改,原因是每个服务器与可执行程序之间传递信息的内容和方式都不尽相同。为此就形成了一个公共标准 CGI,使得为一个服务器写的程序能够在任何服务器上运行。通过这个公共网关接口,服务器可以向 CGI 程序发送信息,CGI 程序也可以向服务器发送信息。可以使用 C Shell、Perl、C、C++、FORTRAN 和数据库语言等任何能够形成可执行程序的语言编写。

如果现在要让 Web 服务器与其他系统结合,比如后台数据库系统,则 CGI 程序会起到程序接口的作用。它将接收到的参数进行预处理,转换成所要结合的数据库系统能够识别的形式。对于数据库系统而言,就是指数据库系统能够识别和运行的标准 SQL 语句。当其他系统完成数据处理后,如果有结果返回,则 CGI 程序可获得并处理其他系统所传回的数据,然后将其按一定的标准格式再送回至 Web 服务器,由 Web 服务器以网页的形式回传到客户端。为了灵活地使用各种数据库系统,CGI 程序支持 ODBC 方式。CGI 程序并不直接访问数据库系统,而是通过 ODBC 数据库接口管理器实现。应用程序以标准 SQL 语句访问 ODBC,由不同的数据库提供的 ODBC 驱动程序将 SQL 语句转换

成数据库所能执行的语言,然后访问数据库。当数据库将结果返回 ODBC 时,ODBC 同样将返回结果进行预处理,以标准形式返回给 CGI 程序。这种使用 ODBC 方式访问数据库的优点是程序员在开发系统时不必考虑后台数据库的类型,只要以标准 SQL 语句编写数据库查询语句访问 ODBC 数据库接口即可,由 ODBC 负责对各种数据库的支持。不论使用大型数据库,还是小型数据库,开发人员都不必更改 CGI 程序。这样就给系统的开发、维护和升级带来很大的方便和灵活性。

13.2.6 ASP 的应用

ASP, 全称 Active Server Page, 它提供了一个在服务器端执行脚本指令的环境(包括 HTML、VBScript 和 JavaScript 等), 通过这种环境, 用户可以创建和运行动态的 Web 应用程序。由于所有的程序都在服务器端执行, 这样就大大减轻了客户端浏览器的负担, 提高了交互速度。利用 ASP 不仅能够产生动态的、交互的高性能的 Web 应用程序, 而且可以进行复杂的数据库操作。ASP 本身包含了 VBScript 和 JavaScript 引擎, 使脚本可以直接嵌入 HTML 中, 而且还可以通过 ActiveX 控件实现更为强大的功能。

确切地说, ASP 并不是一种语言, 它使用的语言通常是 VBScript 或者 JavaScript。通过这两种脚本语言, 我们能够很方便地开发 ASP 应用, 但决不能将 ASP 与 VBScript 或者 JavaScript 等同起来。VBScript 和 JavaScript 之间最大的区别是它们的结构。VBScript 是 Visual Basic 的子集。如果用过 Visual Basic 或者 Visual Basic for Applications(VBA), 就会觉得非常熟悉。不过它们并不是完全一样的, 因为 VBScript 是特意为在浏览器中进行工作而设计的。它不包括脚本范围以外的特性, 如文件访问和打印等。JavaScript 是从一组编程语言如 C、C++ 以及 Java 等脱离出来的。如果以前用过 C 或者 Java, 那么会觉得 JavaScript 的结构非常熟悉。但是, JavaScript 和 Java 是完全不同的两种语言。Java 是一种网页应用程序和非网页应用程序都可以使用的完全成熟的开发语言。而 JavaScript 是一种主要用于脚本编写的脚本语言。

ASP 能够提供六个内建对象, 能够很方便地实现状态保存功能。可以很容易地从客户浏览器获取信息, 并向浏览器反馈信息。这样, 我们就能够很方便地运用 ASP 开发 Web 应用。

ASP 特点有以下几个特点:

(1) ASP 无须编译。ASP 脚本集成于 HTML 中, 无需编译或链接即可直接解释执行。

(2) ASP 易于生成。使用常规文本编辑器即可进行页面的设计。

(3) ASP 独立于浏览器。用户端只要使用可解释常规 HTML 码的浏览器, 即可浏览 ASP 所设计的主页。

(4) ASP 脚本是在站点服务器端执行的。因此,若不通过从服务器下载来观察 ASP 主页,浏览器端将看不到正确的页面内容。

(5) 在 ASP 脚本中可以方便地引用系统组件和 ASP 的内置组件,还能通过定制 ActiveX 服务器组件来扩充功能。与任何 ActiveX Scripting 语言兼容。

(6) 源程序不会外漏。ASP 脚本在服务器上执行,传到用户浏览器的只是 ASP 执行后生成的常规 HTML 码,这样可保证程序代码不会被他人盗取。

ASP 所完成的功能:

(1) 处理由浏览器传送到站点服务器的表单输入。

(2) 访问和编辑服务器端的数据库表。使用浏览器即可输入、更新和删除站点数据库中的数据。

(3) 读写站点服务器的文件,实现访客计数器等功能。

(4) 取得浏览器信息管理等内置功能。

(5) 由 Cookies 读写用户端的硬盘文件,以记录用户的数据。

(6) 可以实现在多个主页间共享信息,以开发复杂的商务站点应用程序。

(7) 使用 VBScript 或 JavaScript 等简易的脚本语言,结合 HTML,快速完成站点的应用程序。通过站点解释器执行脚本语言,产生或更改在客户端执行的脚本语言。

(8) 功能扩充能力强,可通过使用多种程序语言制作的 ActiveX Server Component 以满足自己的特殊需要。

ASP 通过一组统称为 ADO(ActiveX Data Objects)的对象模块来存取数据库,无论采用什么数据库,只要该数据库具有对应的 ODBC 或 OLE DB 驱动程序,ADO 对象就能加以存取。事实上,ASP 提供的 ADO 对象模块包含了下列 6 个对象和 3 个集合,比较常用的是 Connection、Recordset、Command 和 Field 等对象。

Connection 对象: 打开或关闭数据库连接。

Recordset 对象: 存取表的记录,包括读取、插入、删除或更新表的记录。

Fields 集合: Recordset 对象所包含的 Field 对象的集合。

Field 对象: 用来表示表的某一条记录。

Command 对象: 执行查询并返回符合条件的记录(返回值为 Recordset 对象)。

Parameter 集合: Command 对象所包含的参数集合。

Parameter 对象: 用来表示 Command 对象所需要的某个参数。

Errors 集合: 方法调用失败后产生的错误的集合。

Error 对象: 用来表示方法调用失败所产生的某个错误。

ASP 的工作流程说明如下。

当客户端的 Web 浏览器访问某一 Web 站点时,浏览器将 URL 等请求信息发送给

Web 服务器,Web 服务器返回 HTML 响应页面。HTML 页面可以是已经格式化并存储在 Web 节点中的静态页面,也可以是服务器动态创建的以响应用户所提供信息的页面,或者是列出 Web 节点上可用文件和文件夹的页面。

如图 3-10 所示,当用户申请一个 ASP 主页时,Web 服务器响应该 HTTP 请求。当遇到任何与 ActiveX Scripting 兼容的脚本(如 VBScript 和 JavaScript)时,ASP 引擎会调用相应的脚本引擎进行处理。若脚本指令中含有访问数据库的请求,就通过 ODBC 与后台数据库相连,由数据库访问组件 ADO 执行数据库访问操作。ASP 脚本是在服务器端解释执行的,它依据访问数据库后返回的结果集自动生成符合 HTML 语言的主页,以响应用户的请求。所有相关的工作都由 Web 服务器负责。

在结构关系上,ASP 通过 ODBC 与数据库打交道,因此可向上兼容各类数据库系统。而 ASP 产生的 HTML 对客户端的浏览器又有广泛的适应性。但 ASP 对 Web 服务器本身有所挑剔,这看起来似乎是一种缺陷,但实际上也许是一种商业策略——它只支持微软各种操作系统下的 Web 服务器。

图 13-10 表示了 ASP 的工作原理。

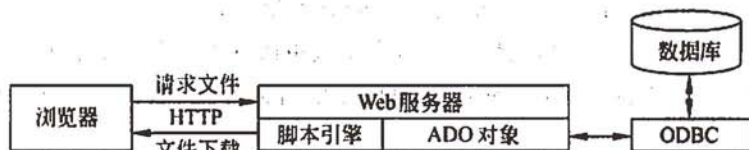


图 13-10 ASP 工作原理图

13.2.7 Servlet 和 JSP 的应用

Servlet 是一 Web 组件程序,它可以动态地生成 Web 内容,支持 Web 应用的 HTTP 协议,使用请求—响应机制。服务器接收请求、处理请求并返回适当的响应。Servlet 采用面向对象的方式对这一过程建模,使之能编写代码处理客户的请求并动态地响应。例如,Servlet 可以从一个表单读取数据并用它更新公司的订单数据库。Servlet 技术在通过动态 HTML 页面扩展 Web Server 上呈现出一种强有力的方法。一个 Servlet 就是一个运行在 Web 服务器中的 Java 程序。Servlet 从浏览器中获取一个 HTTP 请求,动态生成内容(例如查询一个数据库),并把 HTTP 响应信息返回给浏览器。

在 Servlet 之前,CGI 技术被用在动态内容中。然而,由于它的结构以及可升级性的限制,最后证明 CGI 是不太理想的解决方案。

Servlet 技术在可升级性上有了很大的改善,它提供了公认的 Java 平台扩展、安全性以及强壮性等方面的优点。

Servlet 能使用所有标准的 Java APIs。在 Java 领域中,Servlet 技术为密集型应用程序[比如访问一个数据库]提供了很多的优点。优点之一就是 Servlet 运行在服务器端,服务器端具有多种资源且属于一个相对强壮的机器,因此占用客户端的资源相当少。另外一个优点就是 Servlet 在访问数据时更加直接。

JSP 技术由 Sun 公司提出,利用它可以很方便地在页面中生成动态的内容,使网络应用程序可以输出多姿多彩的动态页面。JSP 技术通常与 Java Servlet 技术相结合,可以在 HTML 页面或者其他标记语言中内嵌 Java 代码段并且调用外部 Java 组件。它作为一个前端处理工具,可以使用 JavaBeans 实现复杂的业务逻辑和动态功能。

JSP 代码与 JavaScript 等网页脚本语言是不同的。在标准的 HTML 页面中可以出现的任何内容都可以在 JSP 页面中出现。

在一个典型的数据库应用中,JSP 页面将会调用某些 JavaBean 组件,这些组件可以通过 JDBC 或者 SQLJ 直接或间接地访问数据库。

JSP 页面在运行之前先要解释成 Java Servlet。解释过程是按需进行的,有时也会提前进行。然后它可以处理 HTTP 请求并生成响应信息。JSP 技术为编写 Servlet 程序提供了更为便利的途径。

另外,JSP 页面和 Servlet 程序是可以相互操作的,也就是说 JSP 页面可以包含从 Servlet 程序输出的内容,也可以将内容输出到 Servlet 程序。反过来 Servlet 程序也可以包含从 JSP 页面输出的内容,并且可以将内容输出到 JSP 页面中。JSP 技术的最大优点就是可以将网页的静态内容(HTML 代码)开发与网页的动态内容(Java 代码)开发分隔开来,从而可以使得精通 HTML 但不很精通 Java 的开发人员专门负责网页静态内容的开发,而那些对 Java 很在行但却不熟悉 HTML 的开发人员就可以专注于网页动态内容的开发。

13.3 XML 与数据库

13.3.1 什么是 XML

XML 是 extensible markup language 的缩写,意为可扩展的标记语言。XML 是一套定义语义标记的规则,这些标记将文档分成许多部件并对这些部件加以标识。它也是元标记语言,即可用于定义其他与特定领域有关的、语义的和结构化的标记语言句法的语言。

关于 XML 需要理解的第一件事是,它不仅像超文本标记语言(hypertext markup language, HTML)或格式化的程序那样定义了一套固定的标记,用来描述有限数目的元

索。如果标记语言中没有所需的标记,用户也就没有办法了,只好等待标记语言的下一个版本,希望在新版本中能够包括所需的标记,但这样一来就得依赖于软件开发商的选择了。而 XML 是一种元标记语言,用户可以定义自己需要的标记。这些标记必须根据某些通用的原理来创建,但在标记的意义上,却具有相当的灵活性。例如,假定用户正在处理与家谱有关的事情,需要描述人的出生、死亡、埋葬地、家庭、结婚和离婚等,这就必须创建用于定义每一项的标记。新创建的标记可在文档类型定义(Document Type Definition,在以后的篇幅中常简称为 DTD)中加以描述。XML 定义了一套元句法,与特定领域有关的标记语言都必须遵守。如果一个应用程序可以理解这一元句法,那么它也就自动地能够理解所有由此元语言建立起来的语言。浏览器不必事先了解多种不同的标记语言使用的每个标记。事实是,浏览器在读入文档或它的 DTD 时才了解给定文档使用的标记。关于如何显示这些标记的内容的详细指令是由附加在文档上的另外的样式单提供的。有了 XML 就意味着不必等待浏览器开发商来满足用户的需要了。用户可以创建自己需要的标记,当需要时,告诉浏览器如何显示这些标记就可以了。

关于 XML 需要了解的第二件事是,XML 标记描述的是文档的结构和意义,它不描述页面元素的格式定义。可用样式单为文档增加格式化信息。文档本身只说明文档包括什么标记,而不说明文档看起来是什么样的。XML 是一种元标记语言,可用来设计与特定专业领域有关的标记语言。每种基于 XML 的标记语言都叫做 XML 应用程序。这种应用不像 Web 浏览器或 XML Pro 编辑器那样使用 XML,而是在特定的领域中应用 XML,如化学上用的化学标记语言(chemical markup language,简称为 CML)。每种 XML 应用程序有自己的句法和词汇表,这种句法和词汇表遵守 XML 的基本规则。XML 具有以文本数据为基础的非常灵活的格式。本章讨论的应用都选择 XML 作为基础的原因是(排除大肆宣传的因素),XML 提供了切合实际的、描述清楚的以及易于读写的格式。应用程序将这种格式用于它的数据,就能够将大量的处理细节让几个标准工具和库函数去解决。更进一步地说,应用程序也容易将附加的句法和语义加到 XML 提供的基本结构之上。

13.3.2 XML 文件存储面临的问题

如果采用文件存储 XML,会受到文件系统的限制,出现下面的问题。

1) 大小

第一个局限是文档大小。如果 XML 文件存储了太多的数据,将变得非常不实用。不仅仅因为文档太大,而是维护文档的不同部分变得难于操纵。我们希望处理巨大的文档,并且想检查同其他部分分离的部分文档。

2) 并发性

我们也希望让不同的人在不同的时间更新不同的部分。文件系统中只有一个单一的文档,在同一时间内只有一个人可以处理信息。

3) 工具选择

XML 编辑器可能不是处理同一文档不同部分的合适工具。我们想使用最适合处理数据的工具维护文档的各个部分。

4) 版本

一个经常考虑的重要问题是控制同一文档的不同版本。我们希望能够记录一个文档不同版本的轨迹。

5) 安全

使用不同的工具处理文档的不同部分,并且允许不同的用户在同一时刻处理文档的不同部分也引发出安全问题。我们希望控制一个文档的某一部分只能由某人查看或修改。

6) 综合性:集中和重复

我们希望能够文档中无缝地集成其他的外部数据。

文件系统的局限限制了 XML,而数据库则可以突破文件系统的这些限制,所以将 XML 与数据库相结合是必要的。

13.3.3 XML 与数据库的数据转换

使用 XML 来进行数据传输是很好的方案,因为数据具有高度规范的结构,而 XML 中的那些实体和编码并不重要。毕竟人们关心的仅仅是数据而不是如何在文档中物理地存储这些数据。如果应用程序相对比较简单,关系数据库和数据传输中间件就可以满足需求。如果应用程序庞大而且复杂,那么就需要一个完全支持 XML 的开发环境。

从另一方面来说,假设一个以零散的 XML 文件来组建的网站,不仅需要管理这个网站,还要提供方法使用户可以查询其中的内容,这时网站的文件将非常不规范,而这些文件的使用却变得非常重要,因为这些文件的结构是网站的根本。

要存储或检索数据,可以使用数据库(通常是关系型、面向对象型或者层次型)和中间件(自带或者是采用第三方),也可以使用 XML 服务器(即创建分布式应用平台,如利用 XML 进行数据传输的电子商务应用)。

在选择数据库时,最重要的判断因素可能是利用数据库来保存数据还是保存文档。如果想保存数据,需要的主要是面向数据存储的数据库(例如关系型数据库或者面向对象型数据库)以及在数据库和 XML 文档之间相互转换。

在以数据为中心的文档中的数据内容可能来自数据库(此时想把数据导出为 XML 格式),也可能是 XML 文档(此时想把数据存储于数据库中)。前者的例子是在关系型数

数据库中存储的大量现有数据(或称遗产数据),后者的例子是将数据作为 XML 发布在 Web 中,而且想在数据库中存储以进行更多的处理。如此,根据需求,可能需要将 XML 文档转移到数据库的软件,也可能需要从数据库转移到 XML 文档的软件,或者两者都需要。

将数据存储到数据库中时,经常需要丢弃大量与文档有关的信息,如文档名称。同时还有其物理结构,如实体的定义和使用、属性值和同层元素的顺序、二进制数据的存储方式以及字符数据段和其他编码信息。类似地,当从数据库中检索数据时,生成的 XML 文档除了非预定义的实体,不包含任何字符数据或实体引用。而同层元素和属性的出现顺序也常常就是从数据库中返回的数据的顺序。

这一般是合理的。例如,假设需要用 XML 作为数据格式把一张销售单据从一个数据库中转移到另一个数据库中。在这种情况下,XML 文档中并不关心销售单的编号保存在销售单日期的前面还是后面,也不关心是否将顾客的名称保存在字符数据段还是作为一个外部实体。最重要的在于相关的数据是从第一个数据库转移到第二个数据库中。这样,数据传输软件只需要考虑数据的层次结构(该结构将销售单的有关数据进行分组),其他的则不必过多考虑。

文档的“逆反回归”的不一致效应意指将一个文档的数据存储在数据库中,然后根据这些数据重新组织成新的文档。即便根据标准格式处理,得到的也常常是和前面不同的文档。这是否可以接受要取决于需求,而且也将影响到对数据库和数据传输中间件的选择。

为了在 XML 和数据库之间传输数据,需要在文档结构和数据库结构之间进行相互映射。这样的映射通常分为两大类:模板驱动和模型驱动。

在以模板驱动的映射中,没有预先定义文档结构和数据库结构之间的映射关系,而是使用将命令语句内嵌入模板的方法,让数据传输中间件来处理模板。例如,考虑下面的模板,其中<SelectStmt>元素内嵌了 SELECT 语句:

```
<? xml version="1.0"? >
<FlightInfo>
<Intro>The following flights have available seats:</Intro>
<SelectStmt>SELECT Airline, FltNumber, Depart, Arrive FROM Flights</SelectStmt>
<Conclude> We hope one of these meets your needs</Conclude>
</FlightInfo>
```

当数据传输中间件处理到该文档时,每个 SELECT 语句都将被各自的执行结果所替换,得到下面的 XML 格式:

```
<? xml version="1.0"? >
```



```

<FlightInfo>
<Intro>The following flights have available seats;</Intro>
<Flights>
<Row>
<Airline>ACME</Airline>
<FltNumber>123</FltNumber>
<Depart>Dec 12, 1998 13:43</Depart>
<Arrive>Dec 13, 1998 01:21</Arrive>
</Row>
...
</Flights>
<Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>

```

这种以模板驱动的映射相当灵活。例如,有些产品可以允许在任何结果集合中替换想要的内容(包括在 SELECT 中使用参数);而不是像上面的例子那样简单地格式化结果。另外它还支持编程方法,如增加循环和条件判断结构。还有一些支持 SELECT 语句的参数化,例如通过 HTTP 来传递参数。目前,以模板驱动的映射只支持从一个关系型数据库转换成 XML 文档的情况。

在以模型驱动的映射中,利用 XML 文档结构对应的数据模型显式或隐式地映射成数据库的结构,而且反之亦然。它的缺点是灵活性不够,但是却简单易用。这是因为它是基于具体的数据模型进行映射的,而且能够为用户执行很多地转换工作。由于将数据从数据库转换成 XML 结果依照的是单个模型,因此在这种方式下通常需要结合 XSL 来提供模板驱动的系统所具有的灵活性。XML 文档中的数据视图通常有两种模型:表格模型和特定数据对象模型。有时候也可能会出现其他的模型。例如,通过采用 ID 和 IDREF 属性,一个 XML 文档可以用来表示一个指定的图形。不过,很多现有的中间件并不支持这些模型。

1) 表格模型

许多中间件软件包都采用表格模型在 XML 和关系型数据库之间进行转换。它把 XML 的模型看成一个单独的表格或者一系列表格。也就是说,XML 文档的结构和下面的例子相类似,在单个表格的情况下,<database>并不出现:

```

<database>
<table>
<row>
<column1>...</column1>

```

```
<column2>...</column2>
...
</row>
...
</table>
...
</database>
```

其中的术语“table”可理解为单一的结果集（当从数据库向 XML 中转换数据时），或者是一个单独的表格或可更新的视图（当从 XML 向数据库转换数据时）。如果数据需要来自多个结果集（当数据来自数据库中时），或者与仅仅表达成一系列表格的集合（当转换数据到数据库时）相比，XML 的文档包含更深层次的嵌套元素，那么类似的转换几乎是不可能的。

2) 特定数据对象模型

XML 文档中第二种普遍的数据模型是特定数据对象的树型结构。在该模型中，元素类型通常对应对象，而 XML 中的内容模型、属性和 PCDATA 则对应对象的属性。这种模型直接映射成面向对象的数据库和层次型数据库，当然，借助于传统的对象-关系映射技术和 SQL 对象视图也可以映射成关系数据库。需要注意的是，这种模型并不是文档对象模型(DOM)。DOM 是对文档本身进行建模，而不是对文档中的数据建模。

在 XML 和数据库进行数据转换时，需要考虑许多问题。XML 不支持任何有实际意义的数据类型。所有 XML 文档中的数据都被当成文本来对待，即使它能够用其他的数据类型（如日期或者整数）表示。通常，数据转换中间件将把 XML 文档中的文本转换成其他数据库中的数据类型，反之亦然。然而，特定的数据类型所识别的文本格式是有限制的，如受提供的 JDBC Driver 所支持的数据类型的限制。在这些众多的数据类型中，日期类型通常会导致麻烦。不同国家或地区数字格式的差异也可能产生问题。

在数据库世界中，空值(null)数据意味着数据不存在值。但这与一个值为 0 的数字或长度为 0 的字符串有很大的区别。例如，假设数据来自一个气象站，如果气象站的温度计出了毛病而读不出温度值，那么数据库中 will 存储一个 null 值而不是一个 0。XML 中空值概念的支持可以通过设置可选的元素类型或属性来实现。如果元素类型或属性值为 null，XML 只要在文档中不包含该元素或者属性就可以了。但对数据库而言，空的元素或包含长度为零的字符串属性并不是空值 null：它们的值是长度为 0 的字符串。在 XML 文档和数据库结构之间相互进行映射的过程中，必须特别注意那些可选的元素类型或属性是否对应于数据库中的空值项。如果不这么做的话，很可能出现插入错误（将数据转换到数据库中时）或者无效文档错误（将数据从数据库读出时）。因为同样需要用符

号表示空值,XML 中相对与数据库而言更为灵活。具体来讲,许多 XML 文档很可能包含空字符串的空元素或有一些属性是空值,这个时候必须考虑如何选择合适的中间件来解决这个问题。一些中间件可以让用户选择在 XML 文档中定义用什么来组成空值。除了一些控制字符,XML 文档能够包含任何 Unicode 字符。但许多数据库都限制或不支持 Unicode,而且需要一些特殊的配置才能够处理非 ASCII 编码的字符数据。如果数据包含非 ASCII 字符,那么务必要核实数据库和中间件是否能够处理这些字符。

第 14 章 数据库发展趋势与新技术

14.1 面向对象数据库

数据库技术与面向对象程序设计方法相结合形成了面向对象数据库系统(object oriented database system, OODBS)。

传统的层次、网状和关系数据库系统在许多传统的商业数据库应用中取得了极大的成功,然而在设计和实现更为复杂的数据库应用时,传统数据库系统就暴露出一些缺陷。在设计与实现工程设计和制造数据库、科学实验数据库、电信数据库、地理信息系统数据库以及多媒体数据库的时候,出现了新的应用要求,如长事务的处理,图像或大文本项等新数据类型的存储,以及非标准的特殊应用操作。传统的数据库系统往往不能满足这些复杂数据库应用的要求。

面向对象程序设计方法已经被广泛的应用于软件工程、知识库、人工智能和计算机系统等领域。面向对象程序设计方法和数据库技术的结合,不但能让设计者定义复杂对象的结构,还能让设计者定义作用于这些复杂对象的操作,从而能够有效地支持新一代的数据库应用。

1990 年 7 月,美国高级 DBMS 功能委员会发表了“第三代数据库系统宣言”,提出指导开发第三代数据库系统的 3 条基本原则:

- (1) 第三代数据库系统必须支持数据管理、对象管理和知识管理;
- (2) 第三代数据库系统必须保持或继承第二代数据库系统的技术;
- (3) 第三代数据库系统必须对其他系统开放。

从这个宣言中可知,第三代数据库必须支持面向对象模型。面向对象数据库系统正是以面向对象模型为基础的,它必将成为数据库技术发展的一大趋势。

面向对象数据库产品的研制和开发存在着两大派别,即对象关系数据库和纯粹的面向对象数据库。前者认为关系数据库具有坚实而成熟的理论基础,主张对现有的关系数据库系统进行扩充和改进,使之升级为对象关系数据库系统。具有代表性的对象关系数据库系统产品有 DB2、SMALLTALK/SQL、DBKIT、COMMONBASE、Oracle8 和 SQL Server 7.0 等。纯粹的面向对象数据库派则主张进行彻底的数据库革命,即采用全新的数据模型和模式,抛开现有的数据库系统,从底层做起,使之成为真正的、纯粹的面向对象数据库系统。其代表性的产品有 OBJECTSTORE、ONTOS、VERSANT、IRIS 及

ORION 等。无论是对象关系数据库还是纯粹的面向对象数据库,面向对象的概念和方法是其不可缺少的组成部分。究竟哪一个更适合于存储和访问复杂的数据,具有更优越性的性能,理论界和工业界还有争论,有待于在实际应用中加以比较和检验。

14.1.1 面向对象数据库系统的特征

数据库的特征依赖于实际应用,所设计的数据库语言必须允许用户方便地使用这些特征,数据库的结构也应能有效地支持这些特征。本节结合面向对象的程序设计方法,讨论区别于面向对象数据库系统与传统数据库系统的主要特征。

(1) 面向对象数据库系统应该具有表达和管理对象的能力。面向对象数据库系统通过对象及它们之间的相互联系来描述现实世界。它应该支持对象标识,使得对象的存在不依赖于本身的值,而只依赖于它的标识,对象间能够通过对象标识而相互区分。类的层次和继承是一个关键的概念,新的类允许从以前定义过的类那里继承结构和操作。因此在面向对象数据库系统中,新的对象类型可以简便地重用已有的类型定义。面向对象数据库系统的一个问题是如何表示对象间的联系。ODMG2.0(Object Database Management Group,对象数据库管理组)标准中,提出用一对反向引用来表示二元关系,即把与某个对象相关的对象的标识放在对象内部,并维护参照完整性。

(2) 面向对象数据库系统中的对象可以具有任意复杂度的对象结构。这使对象能够包含所有描述该对象的必要信息。传统数据库与此恰好相反,它把关于复杂对象的信息分散在许多关系或记录中,从而丧失了现实世界的对象与数据库表示之间的直接对应关系。在面向对象数据库系统中,允许逐步细化复杂实体,还能将整个复杂对象或其子集作为一个独立的单位,可以在某一时刻将一成员对象加进去。

(3) 面向对象数据库系统必须具有与面向对象编程语言交互的接口。面向对象程序设计中的对象是瞬态对象,只在程序执行过程中存在。而面向对象数据库可以延长对象的存在,把对象持久地存储起来。对象在程序结束之后仍然持续存在,可以被检索或被其他程序使用。面向对象数据库系统通过与面向对象编程语言交互的接口,可以提供持久化对象和共享对象的能力,从而允许多个程序和应用共享这些对象。

(4) 面向对象数据库应具有表达和管理数据库变化的能力。管理同一对象的多个版本的能力对于设计和工程应用是至关重要的。一个对象的旧版本代表一个已经通过测试和鉴定的设计方案,那么应该保存这个版本直到新版本通过测试和鉴定。除了允许版本变化外,面向对象数据库系统也应该允许模式演变。所谓模式演变是指类的声明发生了变化,或创建了新的类或联系。

综上所述,一个面向对象数据库系统首先应是一个数据库系统,同时又必须具有面向对象的特征。

14.1.2 面向对象数据模型

数据模型是现实世界中的对象或实体,以及对象的约束和对象间的联系的逻辑组织。面向对象数据模型借鉴了面向对象的概念,是面向对象数据库系统所必须支持的数据模型。

面向对象数据库系统是以面向对象数据模型为基础的,是当今数据库技术发展的一大趋势。对于面向对象数据模型,已经有许多基本概念达成了共识,但是仍然缺少一个统一的严格的定义。面向对象数据模型可以看作在一个更高层次上实现数据模型的新成员,并经常用作高层概念模型,尤其在软件工程领域中更是如此。

一系列面向对象的概念构成了面向对象数据模型的基础。概括起来,面向对象数据模型的基本概念有对象、类、继承、对象标识和对象嵌套等,下面将一一加以介绍。

1. 对象结构

我们可以认为一个对象对应着 E-R 模型中的一个实体。对象中封装的属性和方法对外界是不可见的,对象之间的相互作用要通过消息来实现。一般来讲,一个对象有如下相关内容:

(1) 属性集合:一个对象的属性值构成了该对象的状态(类似于关系数据库中关系元组的属性)。属性的值域可以是任何类,包括原子类,如整型值和字符串等。一个属性可以有一个单一值,也可以有一个来自于某个值域的值集,即一个对象的属性可以是一个对象,从而形成嵌套关系。

(2) 方法集合:一个对象的方法可作用于该对象的状态上,同一类对象的所有操作的实现相同。方法的定义规定了方法名称、参数的个数和类型、返回值的类型以及可能的语义描述。方法的实现是一段代码,用来实现方法的功能。方法的定义和实现是相互分离的,为程序员提供了极大的灵活性,甚至可以用不同的语言实现不同的操作。

(3) 消息集合:消息是发送给对象以存取属性值的,除了通过对象所指定的公共界面外,没有其他方法可以访问该对象。对象接收外部传送的消息,执行相应的操作,操作的结果同样可以以消息的形式返回。

2. 对象类

在面向对象数据库中,类是一系列相似对象的集合,对应于 E-R 模型中的实体集概念。类是面向对象系统和数据库系统之间最重要的连接。首先,类直接说明了一个实例及其所属类之间的实例关系;其次,类提供了构成查询的基础;还有,类可以用来增加面向对象数据库的语义完整性;最后,类提出了所有对象的属性和方法的规格说明,便于生成对象。

每个对象是它所在类的一个实例。类的概念类似于关系模式,类的属性类似于关系

模式中的属性。对象类似于元组,类的一个实例对象类似于关系中的一个元组。如果把类本身看作一个对象,则称之为类对象。与其相关的属性集和方法集适用于该类的对象而不适用于该类的实例,这样的属性和方法称之为类属性和类方法。一个类的类属性常常用来描述该类的实例的聚集特性。例如,所有学生实例的“平均年龄”就是一个聚集特性的例子。

3. 继承与多重继承

在面向对象数据模型中,所有类形成了一个有限的层次结构或者是一个有根的无环有向图,我们称之为类层次。如有一个类C和一个连接到C的一组较低层类的集合S,则集合S中的类称为类C的子类,而类C又称为集合S中类的父类。集合S中的任何类继承类C的所有属性和方法,并可以有自己定义的属性和方法。一个父类可以有多个子类,一个子类也可以有多个父类,都存在直接关联或者间接关联的现象。

面向对象数据模型中存在着两种继承:继承(单继承)和多重继承。在大多数情况下,类的继承足以满足应用的要求,典型的树型结构组织用来表示类层次。在树型结构组织中,每个类最多有一个父类,即一个子类只能继承一个父类的属性、方法和消息。然而,有些情况用树型结构并不能很好的表达类层次。多重继承允许一个类从多个直接父类中继承属性、方法和消息,此时的类层次可以用一个有向无环图来表示。

图 14-1 给出了一个学校数据库的类层次结构图,通过下面此图分别解释类层次、继承和多重继承。

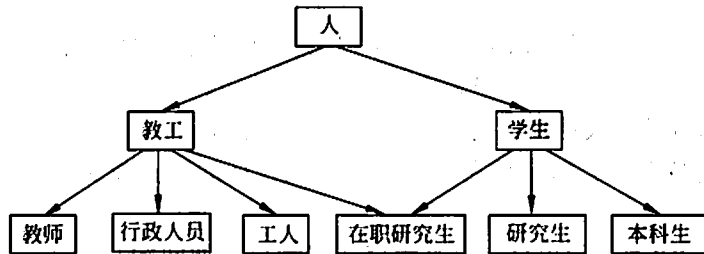


图 14-1 具有多继承的类层次结构图

在这个学校应用的面向对象数据库系统中,“人”是其他所有类的父类,是这个有向无环图的根,是一个最高的类层次;在下面的一个类层次中,教工和学生是人的子类,他们继承了人的所有属性、方法和消息,同时又有本身的特殊属性、方法和消息。在最低的一个类层次中,教师、行政人员、工人和在职研究生是教工的子类,他们继承了教工和人的所有属性、方法和消息。在职研究生、研究生和本科生是学生的子类,他们继承了学生和人的所有属性、方法和消息。值得一提的是,在职研究生既是教工的子类,也是学生的子类,同时继承了教工和学生两个父类的所有属性、方法和消息。

类的继承带来很多优点,子类在继承父类特性的同时,还可以定义自身的属性、方法和消息,但这样就可能和父类的属性、方法和消息发生冲突。这种冲突可能发生在子类和父类之间,通常由系统解决。对于子类和父类之间的同名冲突,一般以子类的定义为准。但在多继承中,一个子类可以有多个父类,如果这些父类中存在同名冲突,就会发生二义性。例如,教工和学生都有“显示信息”方法,它们共同的子类在职研究生就不知道应该继承哪一个方法了。在多继承中有三种处理二义性的方案:一是由用户选择继承的优先次序;二是由系统指定继承某一个父类的定义;三是如果出现了二义性问题,就不允许多继承,甚至有些面向对象数据库系统根本不允许多继承。

4. 对象标识

每个对象都有一个惟一的、由系统生成的对象标识(Object Identifier,OID)。OID 的值对外部用户来说是不可见的,但系统会在内部用这个值惟一地标识每个对象,并用这个值创建和管理内部对象引用。

相对于非面向对象数据模型和程序设计语言来说,对象标识给出了一种更强的标识概念。几种常用的标识形式如下。

(1) 值:用于标识的一个数据值。这种形式的标识常在关系数据库中使用。如一个元组的主码标识了这个元组。

(2) 名称:用于标识的一个由用户提供的名称。在程序设计语言中,用户赋予每个变量一个名字来标识它。在文件系统中,用户给每个文件赋予一个名称来惟一的标识这个文件。

(3) 内置名:以上两种标识是由用户给出的,而内置名则是一种由系统来提供的标识。这种形式的标识在数据模型或程序设计语言中使用。

不同的标识符其持久性程度是不同的,主要有以下几种:

(1) 过程内持久性:标识只在单个过程的执行期间才是持久的,如过程内的局部变量。

(2) 程序内持久性:标识只在单个程序或查询执行期间才是持久的,如程序设计语言中的全局变量、内存指针以及 SQL 语句中的元组标识符。

(3) 程序间持久性:标识在从一个程序的执行到另一个程序的执行期间都保持不变,如指向磁盘上的文件系统数据的指针提供了程序之间的标识,SQL 语句中的关系名也具有程序间持久性。

(4) 永久持久性:标识的持久性不仅仅跨越了各个程序的执行,还跨越了数据结构的重新组织。这种持久性正是面向对象系统所要求的。

对象标识符必须具有永久持久性,也就是说,特定对象一经产生,系统就赋予一个在全系统中惟一的对象标识符,而且是固定不变的,一直到它被删除。面向对象数据库系统

必须具有生成对象标识并维护其永远不变性的机制。

标识符通常是由系统自动生成的,不需要用户来完成这项工作。然而在使用这种功能时要注意,系统生成的标识符通常是特定于这个系统的,如果要数据转移到另一个不同的数据库系统中,则标识符必须进行转化。而且,如果一个实体在建模时已经有一个系统之外的惟一标识符,则系统生成的标识符就可能是多余的。如身份证号码可以作为个人的惟一标识符。

早期的面向对象数据模型要求把所有的一切均表示为对象,无论是一个简单的值还是一个复杂的对象。这将导致这样的情况出现:对于两个整型数值 10 和 20,需要创建两个具有不同 OID 的对象。这种模型需要生成很多的对象标识符,很不实用。因此,大多数面向对象数据库系统允许有对象和值两种表示方法,即每个对象必须有一个永远不变的 OID,但是值没有 OID,值只代表它自己。

5. 对象嵌套

对象嵌套是面向对象数据库系统中的一个重要概念。

在面向对象数据模型中,对象的一个属性可以是一个单一值,也可以是一个来自于值域的值集,即一个对象的属性可以是一个对象,形成嵌套关系,产生一个嵌套层次结构。

一个对象被称之为复杂对象,如果它的某个属性的值是另一个对象。复杂对象主要分为两类:非结构化的复杂对象和结构化的复杂对象。非结构化的复杂对象通常是数据库系统不明确的结构、需要大量存储空间的数据类型,如图像或大文本对象。结构化的复杂对象是指数据库系统清楚对象的内部结构,并可以通过递归生成的对象。

关系模式是对一个二维关系的描述,具有平面的结构。前面讲到的类层次结构形成了对象间的纵向关系,这里的对象嵌套层次结构则形成了对象间的横向关系。我们通过图 14-2 来说明。

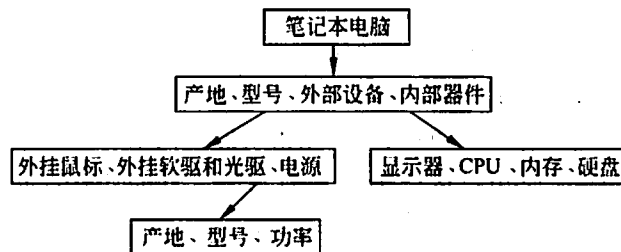


图 14-2 笔记本电脑的嵌套层次

每台笔记本电脑均包括产地、型号、外部设备和内部器件等属性。其中产地和型号的数据类型是字符串,外部设备和内部器件都不是标准数据类型,而是对象。外部设备包括

外挂鼠标、外挂软驱和电源等属性。内部器件包括显示器、CPU、内存和硬盘等属性。电源也是一个对象,包括产地、型号和功率等属性。这样一种嵌套层次结构允许不同的用户采用不同的粒度来观察对象,突出了对象的特征,隐藏了不必要的信息,简化了查询。

14.1.3 面向对象数据库语言

面向对象数据库语言用于描述面向对象数据库的模式,说明并操纵类定义与对象实例。与关系数据库的标准语言 SQL 类似,面向对象数据库语言主要包括对象定义语言和对象操纵语言。对象查询语言是对象操纵语言的一个重要子集。

面向对象数据库语言一般应具备下列功能。

(1) 类的定义和操纵。面向对象数据库语言可以操纵类,包括定义、生成、存取、修改与撤销类。其中类的定义包括定义类的属性、操作特征、继承性与约束等。

(2) 操作/方法的定义。面向对象数据库语言可用于对象操作/方法的定义与实现。在操作实现中,语言的命令可用于操作对象的局部数据结构。对象模型中的封装性允许操作/方法由不同程序设计语言来实现,并且隐藏不同程序设计语言实现的事实。

(3) 对象的操纵。面向对象数据库语言可以用于操纵实例对象。

对象数据库管理组织(Object Database Management Group, ODMG)是面向对象数据库管理系统软件商的国际联盟,曾经提出一种标准,即 ODMG-93 或者 ODMG 1.0。该标准已经修订为 ODMG 2.0。ODMG 对象模型提供了数据类型、类型构造器以及其他一些可以用对象定义语言来说明对象数据库模式的概念,它是对象定义语言和对象查询语言的基础。

对象定义语言被设计为支持 ODMG 2.0 对象模型的语义结构,并且独立于任何特定的编程语言。它的主要用途是创建对象说明,也就是类和接口。因此对象定义语言不是一种完全的编程语言。用户可以独立于任何编程语言在对象定义语言中指定一种数据库模式,然后使用特定的语言绑定来指明如何将对象定义语言结构映射到特定编程语言的结构,如 C++、SMALLTALK 和 Java。

对象查询语言是专门为 ODMG 对象模型指定的查询语言。对象查询语言被设计为能与编程语言紧密配合使用。这些编程语言有一个 ODMG 绑定的定义,如 C++、SMALLTALK 和 Java。这样,嵌入某种编程语言的一个对象查询语言的查询,可以返回与那种语言的类型系统相匹配的对象。对于查询而言,对象查询语言的语法和关系型标准查询语言 SQL 的语法相似,只是增加了有关对象的特征,如对象标识、复杂对象、操作、继承和多态性。

面向对象数据库语言是区别于面向对象数据库与传统数据库的一个重要特征。但面向对象数据库语言的查询功能很弱,这是因为用变量引用对象的方式不能对对象进行统

一管理。如图书作为类,作者是属性。面向对象数据库语言可从每一书的对象找到它的所有作者,但不支持另一方向的查询,即从作者查询他的所有作品,除非把作者也定义成一个类同时把作品设置成属性,这样的设计将在数据库中造成冗余,为此就有必要扩充面向对象数据库语言的查询功能。

数据库系统从网状模型到关系模型的进步使数据库查询语言从用户导航式的过程性语言进入到了由系统自动选择查询路径的非过程性语言。但是非过程性语言的面向集合的操作方式又与高级程序设计语言的面向单个数据的操作方式之间产生了不协调现象,俗称阻抗失配。阻抗失配的根本原因在于关系数据库的数据模型和程序设计语言不一致。因而,对嵌入式数据库语言来说不可避免地产生阻抗失配。但面向对象数据库不同,它的数据模型概念来自面向对象的程序设计方法,因此作为某一面向对象的程序设计语言扩充的面向对象数据库语言,能够从根本上解决阻抗失配问题。

商业的关系型数据库管理系统成功的一个原因在于 SQL 标准。对象数据库管理组织提出的 ODMG 1.0 和 ODMG 2.0 标准包含了对象定义语言和对象操纵语言。但是目前还没有一个关于面向对象数据语言的标准能够像 SQL 标准那样得到业界的普遍支持。现今不同的面向对象数据库管理系统的数据库语言各不相同,要解决这个问题还要花很长的时间。

14.1.4 对象关系数据库系统

在今天的商业领域中,有许多可用的数据库管理系统产品,占统治地位的主要有两个:关系数据库系统和面向对象数据库系统,分别支持关系数据模型和对象数据模型。数据库管理系统产品的另外两种主要类型是层次数据库和网状数据库,它们分别基于层次和网状数据模型。随着数据库技术的发展,后两种数据库系统会逐渐被前两种所取代。

数据库系统面临着许多领域新的应用的挑战,如音频和视频处理系统中的数字化信息,计算机辅助桌面排版系统中的大文本,人造卫星成像或天气预报中的图像,工程设计、生物基因组和建筑图中的复杂数据,股票市场交易历史中的时间序列数据,以及地图数据中的空间和地理数据。显然需要设计某些数据库,它们可以开发、操纵和维护来自这些应用的复杂对象。

在面对上述复杂应用时,基本关系模型及其 SQL 语言的早期版本被证明是不适用的。层次数据模型可以很好地适用于组织中自然存在的分层结构,但它在数据中的内置层次路径上过于局限和固定。网状数据模型可以明确地对联系建模,但在实现方面却需要使用大量的指针,而且不具备对象标识、继承和封装这类概念,也不支持多种数据类型和复杂对象。因此产生了一种趋势,即将对象数据模型中的特征和语言结合到关系数据模型中,这样就扩展了关系数据模型,形成了对象关系数据库系统,使它能够处理当今复

杂的应用。

对象关系数据模型扩展关系数据模型的方式是提供一个包括复杂数据类型和面向对象的更丰富的类型系统。关系查询语言也需要做相应的扩展以处理这些更丰富的类型系统。对象关系数据库系统以对象关系数据模型为基础,为想要使用面向对象特征的关系数据库用户提供了一个方便的迁移途径。

1. 嵌套关系

关系数据理论中定义了第一范式,它要求所有的属性都具有原子的域。原子域指这个域中的元素是不可再分的单元。然而并非所有的应用都用第一范式关系建模最好。例如,某些应用的用户将数据库视为一个对象的集合,而不是一个记录的集合,这些对象可能需要用数条记录来表示。一个简单易用的界面要求用户直观概念上的一个对象与数据库系统概念上的一个数据项之间是一一对应的关系。

嵌套关系模型是关系模型的一个扩展,域可以是原子的也可以赋值为关系。元组在一个属性上的取值可以是一个关系,于是关系可以存储在关系中,从而形成了关系的嵌套。这样,一个复杂的对象就可以用嵌套关系的单个元组来表示。如果我们将嵌套关系的一个元组视为一个数据项,在数据项和用户数据库观念上的对象之间就有了一一对应的关系。

2. 复杂类型

嵌套关系只是基本关系模型扩展的一个实例,其他非原子数据类型,如嵌套记录;同样已被证明是有用的。面向对象数据模型已经导致了对对象的继承和引用等特征的需求。有了复杂对象系统和面向对象,我们能够直接表达 E-R 模型的一些概念,如实体标识、多值属性、一般化和特殊化,而不再需要经过关系模型的复杂转化。

通过对 SQL 的扩展,可以使用复杂类型。下面对有关复杂类型的一些简单概念加以介绍。

下面是一个 books 表的定义:

```
create table books(
...
keyword-set setof (varchar(20))
...
)
```

这个表中的 keyword 属性比较特殊,因为它允许属性是集合。

集合是集合体类型的一个实例,其他的集合体类型包括数组和多重集合。不同于普通关系数据库中表的定义,这里允许属性是集合,E-R 图中的多值属性因而能够直接表述。

现在许多数据库应用需要存储的属性很大,如一个人的照片,或者更大的,如高分辨率的医学图像或者录像剪辑。SQL-99 中提供了新字符型数据大对象数据类型和二进制数据大对象数据类型。大对象一般用于外部的应用,通过 SQL 对它们进行全文检索是毫无意义的。取而代之的是,应用程序一般只检索大对象的定位器,然后用定位器从宿主语言中操作该对象。

下面说明结构类型的声明和使用:

```
create type MyString char varying
create type MyDate
    (day integer,
     month char(10),
     year integer)
create type Document
    (name MyString,
     author-list setof(MyString),
     date MyDate,
     keyword-list setof(MyString))
create table doc of type Document
```

第一个语句定义了一个类型 MyString,它是一个变长的字符串。第二个语句定义了一个类型 MyDate,它有三个组成部分: date、month 和 year。第三个语句定义了一个类型 Document,它包含一个 name、一个作者的集合 author-list、一个类型为 MyDate 的日期以及一个关键词集合。最后创建表 doc,它包含类型为 Document 的元组。上述表的定义与普通关系数据库中表的定义是有区别的,因为前者允许属性为集合或者如 MyDate 那样的具有结构类型的属性,这些特征使得 E-R 图中的复合属性及多值属性能够直接表达。

3. 继承、引用类型

这里的介绍是基于 SQL-99 标准的,不过也会提到一些在这个标准中没有出现的,但在 SQL 标准的未来版本中会引入的一些特征。

继承可以在类型的级别进行,也可以在表的级别上进行。首先考虑类型的继承。

假定我们有如下的人的类型定义:

```
create type Person
    (name varchar(20),
     address varchar(20))
```

再假定需要在数据库中对那些是学生或教师的人分别存储一些额外的信息。由于学

生和教师都是人,因而可以使用类型继承来定义学生和教师类型如下:

```
create type Student
  under Person
  (degree varchar(20),
   department varchar(20))
create type Teacher
  under Person
  (salary integer,
   department varchar(20))
```

Student 和 Teacher 都继承了 Person 的属性,即 name 和 address。Student 和 Teacher 都被称为 Person 的子类型。Person 是 Student 的父类型,同时也是 Teacher 的父类型。

现在假定要存储关于助教的信息,这些助教既是学生又是教师,甚至可能是在不同的系里。如果类型系统支持多重继承,可以为助教定义一个类型如下:

```
create type TeacherAssistant
  under Student, Teacher
```

TeacherAssistant 将继承 Student 和 Teacher 的所有属性,但是却有一个问题,即 name、address 和 department 同时存在于 Student 和 Teacher 中。

name 和 address 属性实际上是从同一个来源即 Person 继承来的,因此同时从 Student 和 Teacher 中继承这两个属性不会引起冲突。然而 department 属性在 Student 和 Teacher 中分别都有定义,另外,一个助教可能是某个系的学生同时又是另一个系的教师。为了避免两次出现的 department 之间的冲突,可以使用 as 子句将它们重新命名,如下面的 TeacherAssistant 类型定义:

```
create type TeacherAssistant
  under Student with (department as student-dept),
  Teacher with (department as teacher-dept)
```

SQL-99 只支持单继承,即一个类型只能继承一种类型,使用的语法如同前面提到的例子。TeacherAssistant 例子中的多重继承在 SQL-99 中是不支持的。

我们通过下面的例子来说明表继承。

假设定义 people 表如下:

```
create table people of Person
```

那么再定义表 students 和 teachers 作为 people 的子表如下:

```
create table students of Student
    under people
create table teachers of Teacher
    under people
```

子表的类型必须是父表类型的子类型,因此 people 中的每一个属性均出现在子表中。

在声明 students 和 teachers 作为 people 的子表时,每一个 students 和 teachers 中出现的元组也隐式地存在于 people 中。所以,如果一个查询用到 people 表,它将查找的不仅仅是直接插入到这个表中的元组,而且还包含插入到它的子表 students 和 teachers 中的元组。然而,只有出现在 people 中的属性才可以被访问。

面向对象的程序设计语言提供了应用对象的能力,类型的一个属性可以是对一个指定类型的对象的引用。我们可以定义一个 Department 类型,它有一个 name 字段和一个引用 Person 类型的 head 字段。然后定义一个 Department 类型的表 departments:

```
create type Department
    (name varchar(20),
    head ref(Person) scope people)
create table departments of Department
```

在上面的定义中,关键词 scope 用于限定引用范围。这里,引用限于 people 表中的元组。

4. 与复杂类型有关的查询

这里要介绍的是处理复杂类型的扩展 SQL 查询语言。与复杂类型有关的查询可以分为如下几类。

1) 路径表达式

在 SQL-99 中,对引用取内容使用→符号。可以使用下面的查询来找出各个部门负责人人的名字和地址:

```
select head→name, head→address
from departments
```

在上面的查询中,带有→符号的表达式被称为路径表达式。

2) 以集合体为值的属性

如果我们想找出所有的码中包含“database”字样的书,可用如下查询:

```
select title
from books
where 'database' in (unnest(keyword-set))
```

unnest(keyword-set)在无嵌套关系的 SQL 中相当于一个 select-from-where 子表

达式。

3) 嵌套与解除嵌套

将一个嵌套关系转换为 1NF 的过程称为解除嵌套。关系 doc 有 author-list 和 keyword-list 两个属性,这两者都是嵌套关系。同时关系 doc 另外还有 name 和 date 两个属性,它们都不是嵌套关系。假定想要将该关系转化为单个平面关系,使其不包含嵌套关系或者结构类型的属性,可以使用以下查询来完成这个任务:

```
select name, A as author, date, day, date, month, date, year, K as keyword
from doc as B, B. author-list as A, B. keyword-list as K
```

from 子句中的变量 B 被声明以 doc 为取值范围,变量 A 被声明以该文档的 author-list 中的作者为取值范围,同时 K 被声明以该文档的 keyword-list 关键词为取值范围。

反向过程(即将一个 1NF 关系转化为嵌套关系)称为嵌套。嵌套可以用对 SQL 分组的扩展来完成。在 SQL 分组的常规使用中,需要对每个组创建一个临时的多重集合关系,然后在这个临时关系上应用聚集函数。如果不用聚集函数而只是返回这个多重集合,我们就可以创建一个嵌套关系。假定有一个 1NF 关系 flat-doc,下面的查询在属性 keyword 上对关系进行了嵌套:

```
select title, author, (date, month, year) as date, set(keyword) as keyword-list
from flat-doc
group by title, author, date
```

5. 函数与过程

对象关系数据库系统中允许用户定义函数与过程,它们既可以用某种数据操纵语言如 SQL 来定义,也可以通过外部的程序设计语言来定义,如 Java、C 或 C++。有些数据库管理系统支持它们自己的过程语言,如 Oracle 中的 PL/SQL 和 Microsoft SQL Server 中的 TransactSQL,它们类似于 SQL 有关过程的部分,但在语法和语义上有所区别,详细信息可参见各自的系统手册。

假设定义这样一个函数:给定一个文档,返回其作者的人数。可以定义这个函数如下:

```
create function author-count(one-doc Document)
return integer as
select count(author-list)
from one-doc
```

其中 Document 是一个类型名。这个函数可用单个文档对象来调用。select 语句同关系 one-doc 一起执行,这个关系仅包括单个元组,即函数的参数。select 语句的结果是单个

值、严格地讲,它是一个只有单个属性的元组,其类型被转化为一个值。

上面的函数可以用于如下查询中,该查询返回具有多个作者的所有文档的名称:

```
select name
from doc
where author-count(doc)>1
```

注意,在上面的 SQL 表达式中,尽管 from 子句中的 doc 指的是一个关系,但在 where 子句中它隐含地被视为一个元组变量,因此可以用来作为 author-count 函数的一个参数。

有些数据库系统允许我们使用如 C 或 C++ 这样的程序设计语言来定义函数。用这种方式定义的函数比用 SQL 定义的函数效率更高,并且能够执行无法用 SQL 完成的计算。使用这些函数的例子很多,如在一个元组的数据上做一个复杂的算法。

用程序设计语言定义的函数需要在数据库系统的外部编译,需要被装入并与数据库系统代码一起执行。这个过程要冒一定的风险,因为程序中的错误可能会破坏数据库的内部结构,并且可能绕过数据库系统的存取控制功能。

使用程序设计语言定义的函数看起来与使用嵌入式 SQL 没什么不一样。使用嵌入式 SQL 时,数据库查询包含在一个通用程序中。但它们之间还是有一个重要差别。在嵌入式 SQL 中,用户程序将查询传送给数据库系统执行,以一次一个元组的形式将结果返回给该程序。因此,用户书写的代码永远不会需要访问数据库本身,于是操作系统就可以保护数据库不被任何用户进程所存取。当在查询中使用用户编码的函数时,要么这些代码必须由数据库系统本身运行,要么该函数所操作的数据必须被拷贝到一个分离的数据空间中。第二种方法增加了系统开销,第一种方法则诱发了潜在的脆弱性,而且同时表现在完整性方面和安全性方面。

6. 面向对象与对象关系

本章研究了建立在持久化程序设计语言上的面向对象数据库,也研究了建立在关系模型之上的面向对象的对象关系数据库。这两种类型的数据库系统市场上都存在,数据库设计者要选择那种适合应用需求的系统。

程序设计语言的持久化扩展和对象关系系统有着不同的市场目标。SQL 语言的声明性特征和有限的能力为防止程序设计错误对数据造成破坏提供了很好的保护,同时使得一些高级优化,如减少 I/O,变得相对简单。对象关系系统的目标在于通过使用复杂数据类型来简化数据建模和查询,典型的应用有复杂数据的存储和查询等。

然而,对于某些类型的应用,如主要在内存中运行和对数据库进行大批量访问的应用来说,一个声明性语言,如 SQL 会带来显著的性能损失。满足应用的高性能要求就是持

久化程序设计语言的目标。持久化程序设计语言提供了对持久数据的低开销存取,并且取消了数据转换要求。但是,持久化程序设计语言对由于程序错误而引起的数据破坏更为敏感,而且通常没有强大的查询能力。其典型应用包括 CAD 数据库。

这些不同种类的数据库系统的能力可以总结如下。

- 关系系统: 简单数据类型、功能强大的查询语言以及高保护性。
- 以持久化程序设计语言为基础的面向对象系统: 复杂数据类型、与程序设计语言集成以及高性能。
- 对象关系系统: 复杂数据类型、功能强大的查询语言以及高保护性。

这些描述具有普遍性,但,请记住,对有些数据库系统来说它们的分界线是模糊的。例如,有些以持久化程序设计语言为基础的面向对象数据库系统是在关系数据库系统之上实现的,这些系统的性能可能比不上那些直接建立在存储系统之上的面向对象数据库系统,但这些系统却提供了关系系统所具有的较强的保护能力。

14.2 ERP 和数据库

14.2.1 ERP 概述

ERP 的形成大致经历了四个阶段: 基本 MRP 阶段、闭环 MRP 阶段、MRP-II 阶段以及 ERP 的形成阶段。ERP 理论的形成是随着产品复杂性的增加,市场竞争的加剧及信息全球化而产生的。

20 世纪 60 年代的制造业为打破“发出订单,然后催办”的计划管理方式,设置了安全库存量,为需求与提前订货提供缓冲。20 世纪 70 年代,企业的管理者们已经清楚地认识到,真正需要的是有效的订单交货日期,因而产生了对物料清单的管理与利用,形成了物料需求计划(MRP)。

20 世纪 80 年代,企业的管理者们又认识到制造业要有一个集成的计划,以解决阻碍生产的各种问题。要以生产与库存控制的集成方法来解决,而不是以库存来弥补或以缓冲时间的方法去补偿,于是 MRP-II,即制造资源计划产生了。

20 世纪 90 年代以来,随着科学技术的进步及其不断向生产与库存控制方面的渗透,解决合理库存与生产控制问题需要处理大量的信息,同时企业资源的管理也更加复杂,因而需要更高的信息处理效率。传统的人工管理方式难以适应以上系统,这时只能依靠计算机系统来实现。信息的集成度要求扩大到企业整个资源的利用和管理,因此产生了新一代的管理理论与计算机系统,即企业资源计划(ERP)。

ERP 是由美国 Garter Group Inc. 咨询公司首先提出的。它是当今国际上先进的企

业管理模式。其主要宗旨是对企业所拥有的人、财、物、信息、时间和空间等综合资源进行综合平衡和优化管理,面向全球市场,协调企业各管理部门,围绕市场导向开展业务活动,使得企业在激烈的市场竞争中能够全方位地发挥足够的能力,从而取得最好的经济效益。下面对 ERP 的形成历史及有关理论和思想分别予以介绍。

1. 基本 MRP

20 世纪 40 年代初期,西方经济学家对库存物料随时间推移而被使用和消耗的规律进行了研究,提出了订货点的方法和理论,并将其运用于企业的库存计划管理中。20 世纪 60 年代中期,美国 IBM 公司的管理专家约瑟夫·奥利佛博士首先提出了独立需求和相关需求的概念,将企业内的物料分成独立需求物料和相关需求物料两种类型,并在此基础上总结出了一种新的管理理论,即物料需求计划(Material Requirements Planning, MRP)理论,也称为基本 MRP。这种理论和方法与传统的库存理论和方法有着明显的不同,其最主要的特点是在传统的基础上引入了时间分段和反映产品结构的物料清单(Bill Of Materials, BOM),从而较好地解决了库存管理和生产控制中的难题,即按时按量得到所需要的物料。基本 MRP 流程如图 14-3 所示。

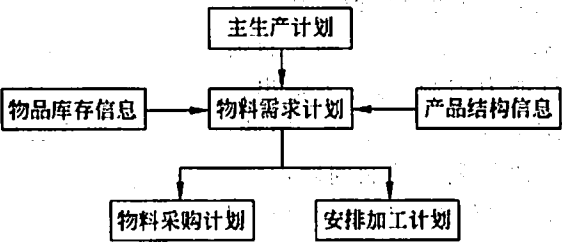


图 14-3 基本 MRP 的流程图

2. 闭环 MRP

在 MRP 的形成及制定过程中,考虑到了产品结构的相关信息和库存的相关信息,但实际生产中的条件是变化的,如企业的制造工艺、生产设备及生产规模都是发展变化的,甚至要受社会环境的影响,如能源的供应和社会福利待遇等。基本 MRP 制定的采购计划可能受供货能力或运输能力的限制而无法保障物料的及时供应。另外,如果制定的生产计划未考虑生产线的能力,因而执行时经常偏离计划,计划的严肃性将受到挑战。因此,利用基本 MRP 原理制定生产计划与采购计划往往不可行。因为信息是单向的,与管理思想不一致:管理信息必须是闭环的信息流,由输入至输出,再循环影响至输入端,从而形成信息回路。因此,随着市场的发展及基本 MRP 的应用与实践,20 世纪 80 年代初在此基础上发展形成了闭环 MRP 理论。

闭环 MRP 理论认为主生产计划与物料需求计划应该是可行的,需要考虑能力的约

束,换言之,即对能力提出需求计划。在能力允许的前提下,才能保证物料需求计划的执行和实现。在这种思想要求下,企业必须对投入与产出进行控制,也就是对企业的能力进行平衡和控制。闭环 MRP 流程如图 14-4 所示。

现在对整个闭环 MRP 的过程进行概述。企业根据发展需要与市场需求来制定企业生产规划,根据生产规划制定主生产计划,同时进行生产能力与负荷的分析。该过程主要是针对关键资源的能力与负荷的分析过程。只有通过对该过程的分析,才能达到主生产计划基本可靠的要求。再根据主生产计划、企业的物料库存信息以及产品结构清单等信息来制定物料需求计划,由物料需求计划、产品生产工艺路线和车间各加工工序能力数据生成对能力的需求计划,通过对各加工工序的能力平衡,调整物料需求计划。如果这个阶段无法平衡能力,还有可能修改主生产计划。采购与车间作业按照平衡能力后的物料需求计划执行,同时进行能力控制,即输入输出控制,并根据作业执行结果反馈到计划层。因此,闭环 MRP 能较好地解决计划与控制问题,是计划理论的一次大飞跃,但它仍未彻底地解决计划与控制问题。

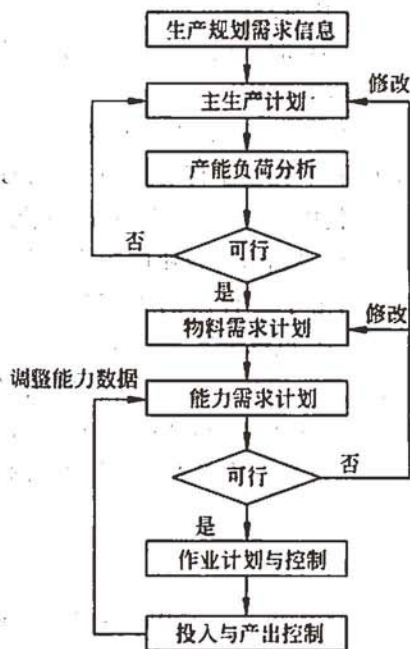


图 14-4 闭环 MRP 的流程图

从闭环 MRP 的管理思想来看,它在生产计划的领域中确实比较先进和实用,生产计划的控制也比较完善。闭环 MRP 的运行过程主要是物流的过程(也有部分信息流),但在实际生产中,产品从原材料的投入到成品的产出,始终伴随着企业资金的流通。对于这一点,闭环 MRP 却无法反映出来。资金的运作会影响到生产的运作,如由于企业资金的短缺,无法按时完成既定的采购计划,这样就会影响整个生产计划的执行。

3. MRP-II

有需求才有发展,市场也是由需求不断推动的。面对新问题的提出,人们就会寻求解决方法。1977 年 9 月,美国著名生产管理专家奥列弗·怀特提出了一个新概念——制造资源计划(Manufacturing Resources Planning),它的简称也是 MRP,但已经是广义的 MRP。为了与基本 MRP 有区别,其名称改为 MRP-II。MRP-II 围绕企业的基本经营目标,以生产计划为主线,对企业制造的各种资源进行统一计划和控制,使企业的物流、信息流和资金流畅通无阻,同时也实现了动态反馈。MRP-II 的逻辑流程图如图 14-5 所示。

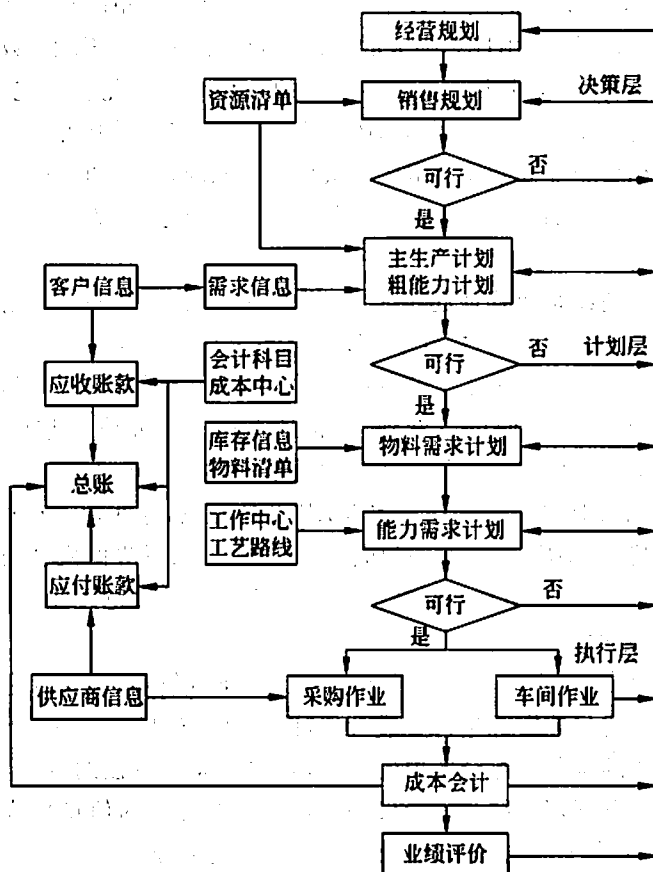


图 14-5 MRP-II 的流程图

下面对不同于闭环 MRP 逻辑流程的部分加以描述。MRP-II 集成了应收、应付、成本及总账的财务管理。其采购作业根据采购单、供应商信息、收货单及入库单形成应付款信息(资金计划)。销售商品后,会根据客户信息、销售订单信息及产品出库单形成应收款信息(资金计划)。可根据采购作业成本、生产作业信息、产品结构信息以及库存领料信息形成生产成本信息。能把应付款信息、应收款信息、生产成本信息和其他信息记入总账。产品的整个制造过程都伴随着资金的流通。通过对企业生产成本和资金运作过程的掌握,调整企业的生产经营规划和生产计划,因而得到更为可行和可靠的生产计划。

MRP-II 理论从 20 世纪 80 年代初开始在企业中得到广泛的应用,MRP-II 的应用与发展给制造业带来了巨大的经济效益。据 1985 年的不完全统计数字,美国有 160 多家计算机软硬件公司,开发与提供了 300 余种 MRP-II 商品软件,已拥有数万家用用户。前西德

也有许多软件公司,开发与提供了数十种商品化的 MRP-Ⅱ 软件。到目前为止,由于 MRP-Ⅱ 所独有的实用性、通用性和强大的生命力及广泛的市场需求,数百个计算机软硬件公司在不同的软硬件环境下开发出功能各异的数百个商品化软件包。根据有关统计,在美国,80%以上的大型企业安装了 MRP-Ⅱ 系统,50%以上的中型企业安装了 MRP-Ⅱ 系统,30%以上的小型企业安装了 MRP-Ⅱ 系统。在德国,95%的大中型企业已应用了计算机系统。在英国,80%的制造业实现了计算机管理。在法国,76%的机械制造企业已应用了计算机管理。

在我国,计算机辅助企业管理起步于 20 世纪 80 年代。1981 年,沈阳鼓风机厂率先引进 IBM 公司的 COPICS 系统,揭开了 MRP-Ⅱ 系统在我国开始应用的序幕。到目前为止,国内已有近 200 家企业引进了十余种国外的 MRP-Ⅱ 软件产品。但是,纵观这些企业 MRP-Ⅱ 系统的应用状况,我们可以看到,真正地全面实施并取得整体效益的企业并不多,其原因主要在于管理模式的差异和实施的质量等方面的问题。

前面讨论了基本 MRP、闭环 MRP 和 MRP-Ⅱ 的理论,这些理论在相应的阶段都发挥了重要的作用,尤其是 MRP-Ⅱ 的发展和应用。MRP-Ⅱ 对世界的发展与应用产生了深远的影响。随着市场竞争日趋激烈和科技的进步,MRP-Ⅱ 思想也逐步显示出其局限性,主要表现在以下几个方面。

(1) 企业竞争范围的扩大,要求企业在各个方面加强管理,并要求企业有更高的信息化集成,要求对企业的整体资源进行集成管理,而不仅仅对制造资源进行集成管理。现代企业都意识到,企业的竞争是综合实力的竞争,要求企业有更强的资金实力,更快的市场响应速度。因此,信息管理系统与理论仅停留在对制造部分的信息集成与理论研究上是远远不够的。与竞争有关的物流、信息及资金要从制造部分扩展到全面质量管理、企业的所有资源(分销资源、人力资源和服务资源等)及市场信息和资源,并且要求能够处理工作流。在这些方面,MRP-Ⅱ 已经无法满足。

(2) 企业规模不断扩大。多集团、多工厂要求协同作战已超出了 MRP-Ⅱ 的管理范围。全球范围内的企业兼并和联合潮流方兴未艾,大型企业集团和跨国集团不断涌现,企业规模越来越大,这就要求集团与集团之间,集团内多工厂之间统一计划,协调生产步骤,汇总信息,调配集团内部资源。这些既要独立,又要统一的资源共享管理是 MRP-Ⅱ 目前无法解决的。

(3) 信息全球化的发展趋势要求企业之间加强信息交流和信息共享。企业之间既是竞争对手,又是合作伙伴。信息管理要求扩大到整个供应链的管理,这些更是 MRP-Ⅱ 所不能解决的。

随着全球信息的飞速发展,尤其是 Internet 的发展与应用,企业与客户、企业与供应商、企业与用户之间,甚至是竞争对手之间都要求对市场信息的快速响应及信息共

享。越来越多的企业之间的业务在互联网上进行,这些都对企业的信息化提出了新的要求。

4. ERP 基本原理、发展趋势

MRP-II 逐步吸收和融合其他先进思想以完善和发展自身的理论。到 20 世纪 90 年代,MRP-II 发展到了一个新的阶段:ERP(Enterprise Resource Planning,企业资源计划)。ERP 系统实现了对整个供应链信息的集成管理,它采用客户机/服务器体系结构和分布式数据处理技术,支持 Internet/Intranet/Extranet、电子商务及电子数据交换。ERP 的逻辑流程图如图 14-6 所示。

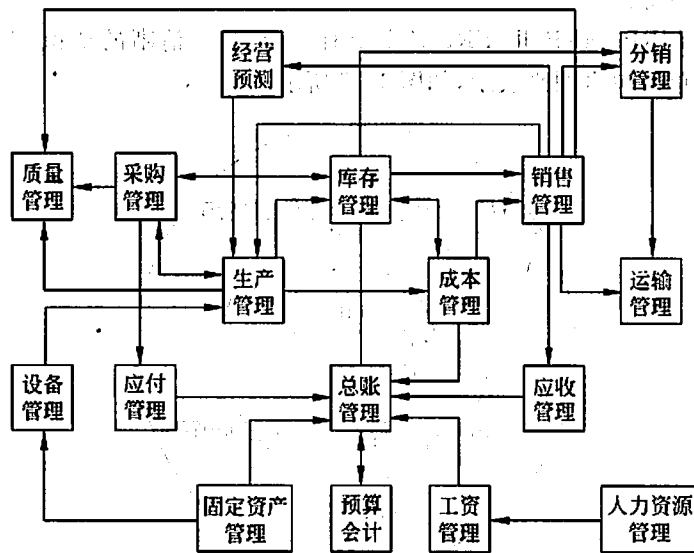


图 14-6 ERP 的流程图

简要地说,企业的所有资源包括三大流:物流、资金流和信息流。ERP 也就是对这 3 种资源进行全面集成管理的管理信息系统。概括地说,ERP 是建立在信息技术基础上,利用现代企业的先进管理思想,全面集成了企业的所有资源信息,并为企业提供决策、计划、控制与经营业绩评估的全方位和系统化的管理。ERP 系统是一种管理理论和管理思想,而不仅仅是信息系统。它利用企业的所有资源,包括内部资源与外部市场资源,为企业制造产品和服务创造最优的解决方案,最终达到企业的经营目标。由于这种管理思想必须依附于电脑软件系统的运行,所以人们常把 ERP 系统当成一种软件,这是一种误解。要想理解与应用 ERP 系统,必须了解 ERP 的实际管理思想和理念。

ERP 理论与系统是从 MRP-II 发展而来的,它除继承了 MRP-II 的基本思想(制造、

供销及财务)外,还大大地扩展了管理模块,如多工厂管理、质量管理、设备管理、运输管理、分销资源管理、过程控制接口、数据采集接口以及电子通信等模块。它融合了离散型生产和流程型生产的特点,扩大了管理的范围,更加灵活地开展业务活动,实时地响应市场需求。它还融合了多种现代管理思想,进一步提高了企业的管理水平和竞争力。因此,ERP理论不是对MRP-II理论的否认,而是继承与发展。MRP-II的核心是物流,主线是计划,伴随着物流的过程,同时存在资金流和信息流。ERP的主线也是计划,但ERP已将管理的重心转移到财务上,在企业整个经营运作过程中贯穿了财务成本控制的概念。总之,ERP极大地扩展了业务管理的范围及深度,包括质量、设备、分销、运输、多工厂管理和数据采集接口等。

为了便于对MRP/MRP-II/ERP各系统有一个完整、清晰的认识,下面将它们之间的关系用一个简单的包含图来表示,如图14-7所示。

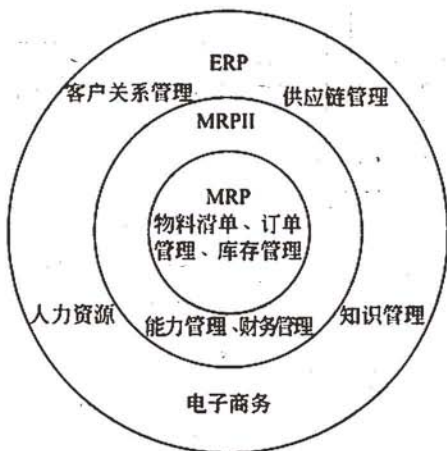


图 14-7 MRP/MRP-II/ERP 关系图

1990年,Garter Group公司率先提出了ERP的概念。10年之后,该公司又提出了一个新的概念——ERP-II。ERP从诞生之日起就在不断发展,这里姑且先不讨论ERP-II这个名称叫法。下面从几个方面对ERP未来的发展趋势进行展望。

1) 管理范围更加扩大

ERP的管理范围有继续扩大的趋势,继续扩充供应链管理(Supply Chain Management,SCM),SCM融合了企业本身的所有经营业务、企业的办公业务以及企业之间的协同商务业务等,如电子商务、客户关系管理和办公室自动化等。协同商务是指企业内部人员贯穿于贸易共同体的业务伙伴和客户之间的协作,及电子化的业务交互过程。贸易共同体可以是一个行业或行业分支,也可以是供应链或供应链的一部分。此外,ERP

系统还日益和计算机辅助设计、计算机辅助工艺设计、产品数据管理、POS 系统以及自动货仓等系统融合。这样,企业管理人员在办公室中完成的全部业务都被纳入管理范围内,实现了对企业的所有工作及相关的内外环境的全面管理。

2) 继续支持与扩展企业的流程重组

企业的外部与内部环境变化是相当快的。企业要适应这种快节奏的变化,就要不断地调整组织机构和业务流程。因此,ERP 的发展必然要继续支持企业的这种变化,使企业的工作流程能够按照业务的要求进行组织,以便集中相关业务人员,用最少的环节,最快的速度 and 最经济的形式,完成业务的处理过程。

3) 运用最先进的计算机技术

信息是企业管理和决策的依据,计算机系统能够及时而准确地为企业提供必要的信息,因此 ERP 的发展是离不开先进的计算机技术的。Internet 和 Intranet 技术使企业内部及企业与企业之间的信息传递更加畅通。面向对象技术的发展使企业内部的重组变得更加快捷和容易。计算机在整个业务过程中产生的详尽信息记录与统计分析,使决策变得更加科学和有目的性。新的计算机技术的不断涌现为 ERP 的发展提供了广阔的前景。

ERP 系统是高度集成的信息管理系统,并且要在实践中不断积累与应用验证,这就给软件的设计、开发和维护工作带来了很大困难,它的开发与单一模块的软件开发有着很大的区别。企业各个业务之间错综复杂的联系是 ERP 系统的软件设计和开发工作必须攻克的难关。

5. ERP 设计的总体思路

ERP 设计的总体思路即把握一个中心、两类业务和三条干线。

企业的主要目的是赢利,因而企业的每个业务活动都要考虑企业的经营目标,都会有输入的费用和输出的业务结果。因此,各项业务活动和功能模块都要考虑归集到财务数据。财务应是各项业务的归集中心,这是在系统规划与设计实现时必须考虑到的。同时财务的处理要考虑本国国情,使从其他模块传递到财务的数据符合财务制度要求,可以为财务所利用;保证各个模块与财务之间数据传递的有效和畅通,实现财务业务处理的高度集成。

从 ERP 原理可以得出这样一个结论:计划与执行贯穿了系统的整个过程。从计划到执行计划,再反馈到计划层,影响计划的制定和修正,这个过程周而复始,形成一个闭环,也体现了管理的闭环原则。各个模块的业务处理,围绕计划展开,计划有经营规划、销售计划、主生产计划、采购计划、资金需求计划以及车间作业计划(生产和检验)等。

ERP 设计的三条干线为供应链管理、生产管理和财务管理。这三条干线也就是制造业业务处理的主流业务,因而在进行设计规划与设计实现时要围绕这三条干线进行分工和协调。其过程分述如下。

(1) 供应链管理是企业物流业务的主干线,它处理企业从原材料供应,产品存储到产品销售的整个流程。其中物流管理的核心是库存管理,并要综合考虑整个物流供应链的管理。该过程的主要信息和数据有物品代码资料、物品库存资料和供需双方的资料等,其中销售计划、合同和定单是主生产计划的入口数据。

(2) 生产业务是制造业的主体业务,包含主生产计划的制定、资源的利用、下达生产计划和生产作业的控制等业务。这个过程的运作涉及很多企业的重要基础数据,如产品结构清单、工艺路线、工作中心资源与能力等,在设计时要尽可能地考虑各个行业的不同产品结构特点、工艺特点和业务管理特点等。

(3) 财务管理。财务集成设计是最终完成 ERP 集成的关键,是企业各项业务活动最终结果的体现,也是一个中心的最终体现。

这三条干线的数据相互利用,业务互相联系与渗透。因此,做好数据库的设计非常重要,它直接关系到集成的好坏和系统效率的高低。设计 ERP 数据库时应尽量将一个实体的描述放入一个数据库中,如在设计基础数据采集入口时,规定一种数据只能从一个入口录入,其他地方只是使用数据或继续补充,不再重复录入。这样就可以避免系统中对一个事件的描述存在数据不一致的问题,达到了数据只录入一次然后充分共享的目的。既减少了录入数据的工作量,又降低了出错的几率。利用数据库的约束规则可以做到这一点,这涉及数据库数据处理的效率问题。

围绕这三条干线的模块划分如下。

(1) 进销存管理模块系列,包括库存管理、销售管理、采购管理及分销资源计划管理等。

(2) 生产管理模块系列,包括制造标准、主生产计划、物料需求计划、能力需求计划、车间作业管理、重复制造生产管理、质量管理及设备管理等。

(3) 财务管理模块系列,包括总账管理、应收账款管理、应付账款管理、预算会计、现金管理、账簿报表管理、固定资产管理、工资管理及成本会计等。

(4) 其他补充模块,如人力资源管理、技术管理、经营预测系统、决策系统和工作流管理等。

14.2.2 ERP 与数据库

在 ERP 软件中,数据库是它的灵魂。ERP 就是结合企业已有的数据,更好地管理它们在金融、需求、物流、人力资源和订货方面的经营活动。其目的是降低成本、压缩运送时间、减少冗员和存货,以及改进客户服务。为了达到提高利润的目的,公司首先必须从其客户和销售活动中积累信息,并把它们存入一个不断更新的数据库。

数据库营销的基本原则非常简单。如果营销活动是由客户驱动而不是由产品驱动

的,那么营销的效率会更高。公司现在和过去的客户资料对于设计未来产品和市场营销计划都是非常宝贵的。公司的业务前景很大一部分将取决于已经购买其产品的客户。品牌忠诚度将越来越重要。树立品牌忠诚度要靠令人满意的产品,看得见的可靠服务,以及企业和客户之间卓有成效的双向沟通。

应该牢记,客户数据库在企业的主要用途方面是大同小异的。若不出现严重的错误,通过适当的构想和执行,数据库营销将给企业带来客户忠诚度、重复销售、降低成本、交叉销售以及企业前景的高度认同。万一发生了这种错误,数据库营销将会失败,其失败原因如下:

- (1) 缺乏营销计划;
- (2) 把注意力集中在价格上而不是服务上;
- (3) 公司内部资源还不足以建立数据库;
- (4) 没有长期的营销计划;
- (5) 计算机成本过高;
- (6) 数据没有跟进和更新;
- (7) 对结果没有跟踪;
- (8) 缺乏完善的领导,数据库无法运作。

ERP 为工作完成量和资金周转率提供了许多新的方法,如电子交易或客户关系管理。之所以成功的一个重要因素是在企业的现有水平上创造和应用共同的数据。企业中的每个人必须以统一的语言和模板,登录和充实数据库信息,也就是说,创建一个标准化的商业流程。这同样也意味着企业必须建成一个不断发展变化的组织,允许人们改变他们的工作方式。ERP 给企业员工注入了一种持续改进的思维模式。在企业内部通过支持一种创新与拓展机会的网络,使员工承担变革的责任,这就是目前 ERP 向所有寻求发展的企业所提出的重大挑战。

ERP 系统可以是传统的 C/S 结构,也可以是现在流行的 B/S 模式的三层结构。前台使用的开发环境和后台的数据库也不尽相同。在实际的 ERP 系统开发中,经常要同时对多个数据库操作,一般是一个主表带多个从表,主表的某一个字段和从表的某一字段关联,而主表该字段的数据要从对应从表中获取,多表操作可以归纳为主从表操作。

每一个 ERP 软件都有自己的数据库,而这些数据库中最关键的是数据库框架。在编写 ERP 数据库的应用程序时,首先要做的一件事就是建立数据库框架,至少包括数据库和数据库中的表,当然还有视图和存储过程等。然后使用编程语言开发用户界面,接受用户对数据库的操作。当成功地开发了一个 ERP 软件后,需要将它打包,最后交给客户安装并使用。这时就有一个问题,打包时,不能将数据库管理系统打包到安装程序中。在用

户使用 ERP 软件之前必须建立好数据库框架,但用户并不知道数据库的框架结构,而 ERP 软件又必须访问特定的数据库框架才能成功运行,这时我们就需要有一个可以自动生成数据库框架的程序。例如,当开发一个人力资源管理系统时,需要一个数据库框架,数据库中至少应包含一个表,表中包含姓名、年龄和工资等信息,然后通过客户端访问这个表。如果没有这个表,程序就不可能成功地运行。

现在的 ERP 软件中都具有自动生成数据库框架的功能。不同软件的实现方法也不同,总结一下,大约有 3 种:

- (1) 使用向导自动生成数据库框架;
- (2) 在安装时配置数据库系统;
- (3) 集成在主程序中,当主程序第一次运行的时候自动生成数据库框架。

不论采用哪种实现方法,它们的用途都是一样的。

下面我们结合 ERP 系统的设计过程说明在 ERP 系统中对数据库技术的应用。

首先,系统分析员要深入理解 ERP 的原理,进一步系统地分析与描述各个管理模块的业务处理要求与处理过程,同时要结合企业和行业自身的特点,进一步细化分析行业的特殊业务要求,绘制出其管理子系统的业务处理流程图。

其次,在详细分析与理解相应管理业务的基础上绘制出数据流程图。系统分析员根据企业管理的具体情况或软件定位的行业范围,对数据流图进行细化和扩展,最后定义数据字典,为数据库设计做好准备。

然后,结合业务流程与数据流程图,利用数据流程图的分析结果,使用 E-R 图进一步详细分析各个实体的属性,为进一步的数据库设计打下基础。

最后,根据 ERP 原理分析出功能模块图。同时从管理业务与软件运行两个角度进行综合考虑,得出一般的主流功能模块。这并不说明 ERP 的各个业务只具有这些功能模块,不同的行业运用,不同的管理实情,可以设计出不同的功能模块。并且,如果选用不同的开发语言和设计技术,各个功能模块的集成度与模块的划分也会有所不同。

综上所述,数据库及其技术是实现 ERP 系统的基础。

14.2.3 案例分析

早期的 MRP 是基于物料库存计划管理的生产管理系统。MRP 系统的目标是:围绕所要生产的产品,应当在正确的时间,正确的地点,按照规定的数量得到真正需要的物料。按照各种物料真正需要的时间来确定订货与生产日期,以避免造成库存积压。

本节将讨论一个库存管理系统的设计。为了突出基本 MRP、闭环 MRP、MRP-II 和 ERP 这一系列系统和数据库的关系,这个系统将设计为 MRP 系统,而不是 ERP 系统。这可以避免把内容集中在繁杂的企业业务流程上,又说明了系统与数据库之间的

关系。

一个库存管理系统中的业务主要有以下几种。

1) 物料的出入库、移动管理

对日常的生产领料、销售提货、采购入库、生产入库和物料的库位移动等工作进行管理,产生出入库和移动单据,改变仓库和货位的库存数量,登记物品数量。

2) 库存物料定期盘点,调整物料的库存量,做到账物相符

根据物料的盘点周期对每一种库存物料进行盘点,并按照实盘数量调整物料库存数量。盘点方法一般有冻结盘点法和循环盘点法两种。冻结盘点需要对物料停止进行入库和出库操作。而循环盘点则可以同时进行入库和出库处理,盘点结果形成盘点报表,经过财务审核确认后用以调整库存数量。

3) 库存物料管理信息分析

从各种角度对库存物料信息进行分析。如库存物料数量分析(是否超储或短缺),物料占用资金分析、物料来源和去向分析以及物料分类构成分析等。

因此,库存管理子系统的主要功能设计应为:对生产作业或其他物料领用进行管理,编制领料单,并凭单发货。对仓库的日常操作,如入库、出库和调拨等业务处理进行管理,并编制有关出、入库单据,同时凭单记录库存账目。按物料的盘点周期进行盘点和清查工作,编制盘点报表,上报给财务部门,审批后按实盘量调整库存。

库存管理业务数据流程图可以分为3层。图14-8是企业库存管理业务的第一层数据流图,库存业务管理被分成基本数据管理和库存业务管理两个子系统。

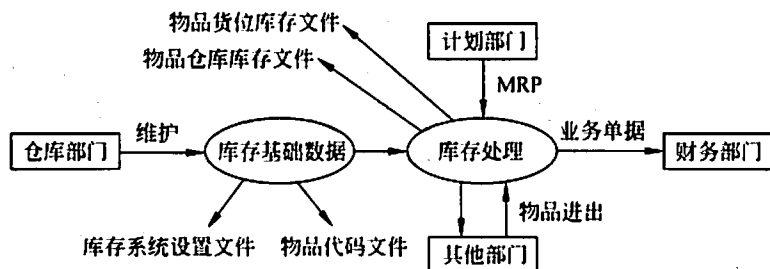


图 14-8 企业库存管理第一层数据流图

继续对图14-8中的两个子系统进行分解,得到库存管理业务的第二层数据流图如图14-9所示。

继续对图14-9中的入库和出库子系统进行分解,得到库存管理业务的第三层数据流图如图14-10、14-11和图14-12所示。

根据企业库存管理各层业务数据流图的详尽描述,经过加工处理,设计出库存管理系

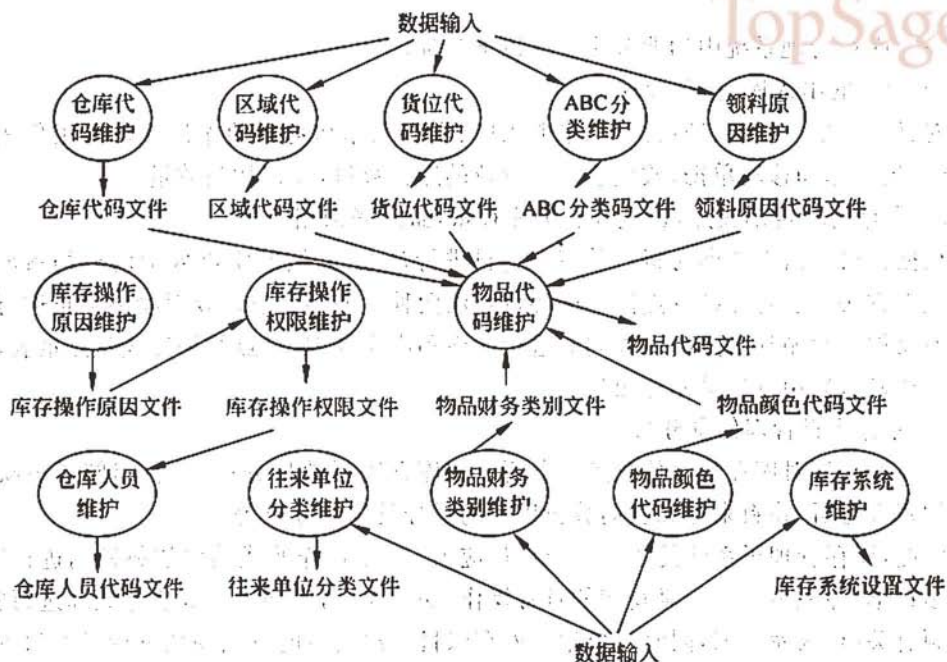


图 14-9 库存基础数据管理数据流图

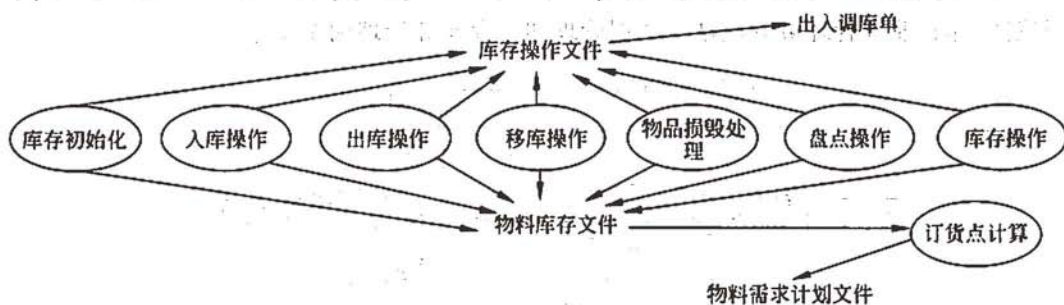


图 14-10 库存处理数据流图

统的 E-R 关系图如图 14-13 所示。

根据库存管理业务流程和数据流图,可以对系统的模块功能进行设计,设计出库存管理系统的功能模块。参见图 14-14。

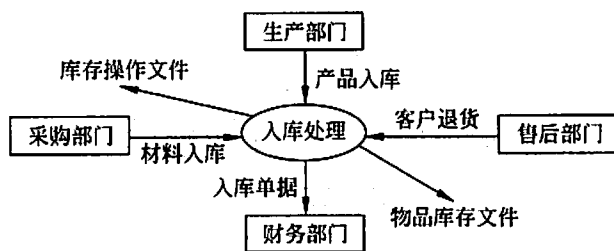


图 14-11 入库处理展开数据流图

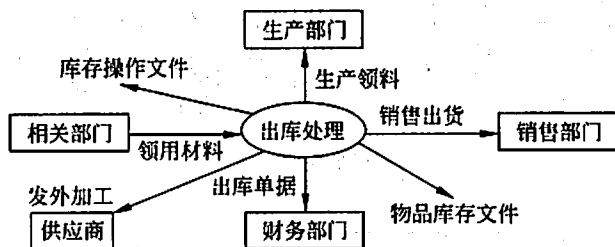


图 14-12 出库处理展开数据流图

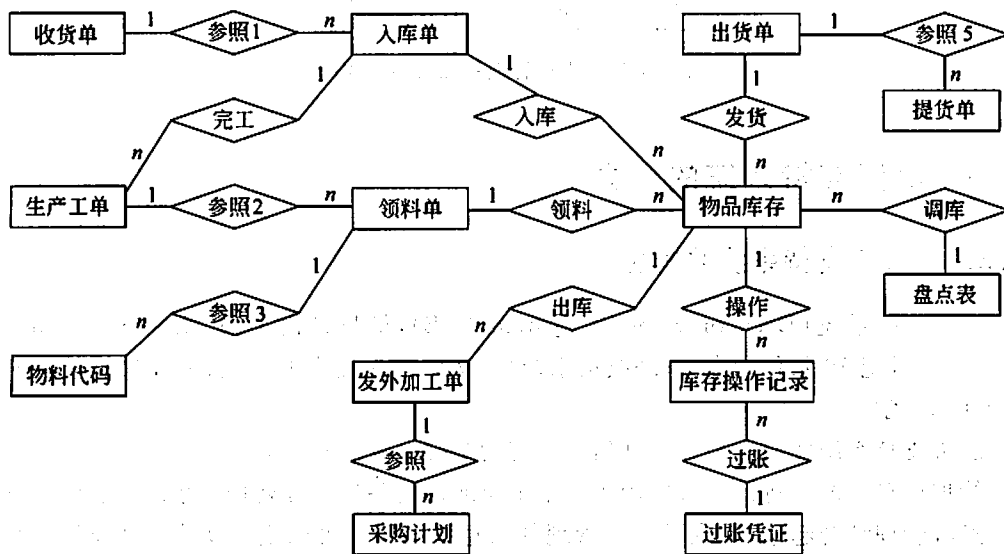


图 14-13 企业库存管理 E-R 关系图

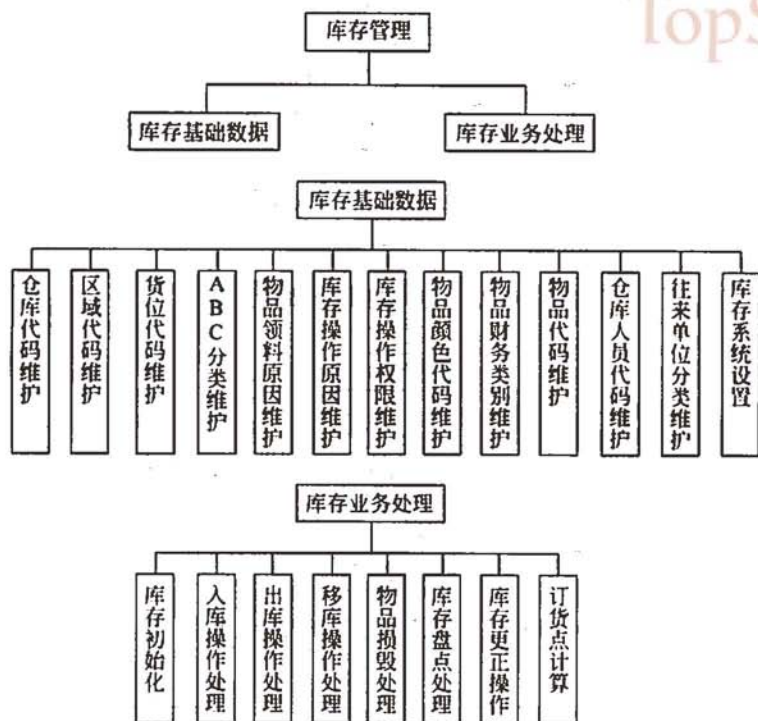


图 14-14 库存管理系统功能模块图

14.3 决策支持系统的建立

14.3.1 决策支持系统的概念

决策支持系统(Decision Support System, DSS),最早是由美国 M. S. Scott Morton 教授于 20 世纪 70 年代初在《管理决策系统》一文中首先提出的,是目前发展比较迅速的新型计算机学科。

决策支持系统实质上是在管理信息系统的基础上发展起来的。

管理信息系统是将计算机应用于一个单位或部门各种业务处理的系统。它将数据处理和经济管理结合起来,形成了用于管理的信息系统。它可以定义为:管理信息系统是一个人、机结合的对管理信息进行收集、传递、存储、加工、维护和使用的系统。

管理信息系统可简单地理解为:

管理信息系统 = 管理业务 + 数据库技术

各种类型的管理业务主要以账本和处理过程为核心。数据库技术主要是数据库系统和数据处理程序。数据库系统是由数据库管理系统和数据库组成的。数据处理程序是用数据库语言对管理功能中的处理过程编写出的程序。数据库语言是一种数据处理语言,它具有对数据库的建立、增加、删除、更新、查询、统计、恢复和报表等功能,以及计算机语言的基本结构,如顺序、选择和循环。管理功能中的账本数据按数据库结构形式存入数据库中。

管理信息系统能够使各企事业单位的管理由原来的人工处理变成由计算机处理大量繁琐事务的科学管理,从而使管理提高到新的水平。管理信息系统的最大优点是它能对大量的数据进行有效的管理和处理。它的局限在于对管理者提供的辅助决策信息只表现为数据的查询和统计形式。

通过引入数学模型,我们可以增强对复杂问题的处理能力,使人们尽可能地按客观规律办事,避免错误,取得预期的效果。这样的系统称之为模型辅助决策系统。

模型是辅助决策的重要手段,是专家学者在探索事物的变化规律中抽象出的数学模型。这项工作是创造性劳动,需要花费大量的精力和敏感思维来得到规律性或相近的数学模型。数学模型建立后的一个重要问题就是找出该模型的求解算法。可以是精确求解,也可以是近似求解。有了模型算法,我们就可以用计算机语言来编制成程序。实际的决策者就可以在计算机上执行模型程序,计算出结果,得到辅助决策信息。

模型辅助决策系统一般是用单个典型模型来解决某一类的决策问题。随着管理信息系统的发展,在管理信息系统中除完成大量的数据组织、存储、查询、统计和报表生成等主要工作外,还要逐步增加模型辅助决策的功能。

随着新技术的发展,所需要解决的问题愈来愈复杂,除了完成一般管理信息系统的要求外,所涉及的模型愈来愈多,不仅是几个,而是十多个、几十个甚至上百个模型来解决同一个问题。对于这样的问题,使用管理信息系统或者模型辅助决策系统都不能解决,需要应用决策支持系统。决策支持系统的新特点就是增加了模型库和模型库管理系统。它把众多的模型有效地组织和存储起来,把模型库和数据库有机地结合起来。决策支持系统的基本结构如图 14-15 所示。

综上所述,决策支持系统的定义为:综合利用各种数据、信息和知识,特别是模型技术,辅助各级决策者解决决策问题的人机交互系统。

决策支持系统由下列子系统组成。

(1) 数据库子系统。包括数据库及有关决策问题的数据,并由数据库管理系统进行管理。

(2) 模型库子系统。包括模型库,其中有财务、统计、管理科学或其他定量模型,可提供系统的分析功能,由模型库管理系统为用户提供建模语言以及模型库管理功能。

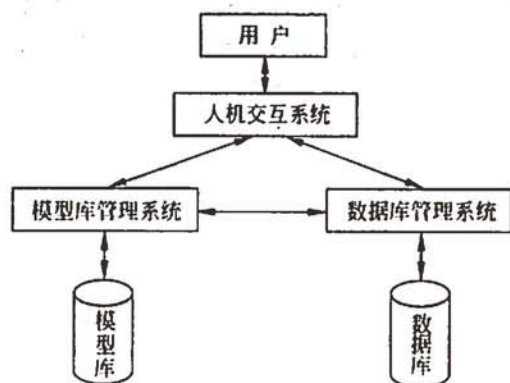


图 14-15 决策支持系统的构成

(3) 人机交互系统。通过该子系统,用户与决策支持系统通信并使用决策支持系统,协调和控制数据库子系统和模型库子系统的管理和运行。

(4) 用户。用户也是系统的一部分,研究人员认为用户与计算机的频繁对话可以对决策支持系统产生某些特殊的作用。

在决策支持系统基本结构的基础上,增加知识库子系统,形成智能决策支持系统结构。该结构中的知识库子系统包括知识库和推理机。知识库子系统能支持其他子系统或作为独立的部件应用,提供智能和定性分析功能,以增强决策人的能力。

14.3.2 数据仓库设计

由于数据库主要面向日常事务处理,不适合进行分析处理。因此一种新的技术应运而生,这就是数据仓库技术。数据仓库技术是公认的信息利用的最佳解决方案,它不仅能够从容解决信息技术人员面临的问题,同时也为商业用户提供了很好的商业契机。

数据仓库已成为建立决策支持系统的重要技术手段,是建立决策支持系统的基础。在这一节里,我们将讨论数据仓库的设计方法,建设数据仓库的三级数据模型,如何提高数据仓库的物理性能,以及数据仓库的元数据等有关内容。

数据仓库的数据具有四个基本特征:面向主题的、集成的、不可更新的以及随时间不断变化的。这些特点说明了数据仓库从数据组织到面向分析的数据处理都与原来的数据库有较大区别,也决定了我们在进行数据仓库系统设计时,不能够照搬传统的数据库系统开发方法,因而需要寻找一个适于数据仓库设计的方法。

所谓数据模型,就是对现实世界进行抽象的工具,抽象的程度不同,也就形成了不同抽象级别层次上的数据模型。数据仓库的数据模型与操作型数据库的三级数据模型又有

一定的区别,主要表现在:

- (1) 数据仓库的数据模型中不包含纯操作型的数据。
- (2) 数据仓库的数据模型扩充了码结构,增加了时间属性作为码的一部分。
- (3) 数据仓库的数据模型中增加了一些导出数据。

可以看出,上述3点差别也就是操作型环境中的数据与数据仓库中的数据之间的差别,同样也是数据仓库能够对数据进行分析处理所要求的。虽然存在这样的差别,在数据仓库设计中,仍然存在着三级数据模型,即概念模型、逻辑模型和物理模型。

概念模型是主观与客观之间的桥梁,它是一个概念性的工具,用于设计系统,收集信息。具体到计算机系统来说,概念模型是客观世界到机器世界的一个中间层次。人们首先将现实世界抽象为信息世界,然后将信息世界转化为机器世界。信息世界中的这一信息结构,即我们所说的概念模型。

概念模型最常用的表示方法是使用 E-R 图作为它的描述工具。E-R 图描述的是实体以及实体之间的联系。在 E-R 图中,长方形表示实体,在数据仓库中就表示主题,在长方形内写上主题名;椭圆形表示主题的属性,并用无向边把主题与其属性连接起来;用菱形表示主题之间的联系,菱形框内写上联系的名字。用无向边把菱形分别与有关的主题连接,给无向边标记上联系的类型。若主题之间的联系也具有属性,则把属性和菱形也用无向边连接上。

由于 E-R 图具有良好的可操作性,形式简单,易于理解,便于与用户交流,对客观世界的描述能力也较强,在数据库设计方面得到了广泛的应用。因为目前的数据仓库一般都建立在关系数据库的基础之上,为了和原有数据库的概念模型保持一致,采用 E-R 图作为数据仓库的概念模型仍然是较为合适的。

在数据仓库设计中采用的逻辑模型就是关系模型。无论是主题还是主题之间的联系,都用关系来标识。关系模型概念简单清晰,用户易懂易用,有严格的数学基础和在此基础上发展的关系数据理论。关系模型简化了程序员的工作和数据仓库设计开发的工作,当前比较成熟的商品化数据库产品都是基于关系模型的。因此采用关系模型作为数据仓库的逻辑模型是合适的。数据仓库的逻辑模型描述了数据仓库的主题的逻辑实现,即每个主题所对应的关系表的关系模式的定义。

所谓数据仓库的物理模型就是逻辑模型在数据仓库中的实现,如物理存取方式、数据存储结构、数据存放位置以及存储分配。物理模型是在逻辑模型的基础上实现的。在进行物理模型设计实现时,应考虑的主要因素有 I/O 存取时间、空间利用率和维护代价。在进行数据仓库的物理模型设计时,考虑到数据仓库的数据量大,但操作单一的特点,可采取其他一些能够提高数据仓库性能的技术,如合并表、建立数据序列、引入冗余、进一步细分数据、生成导出数据以及建立广义索引等。

在建立数据仓库过程中,一个重要问题是如何提高系统的性能。因为数据仓库的数据量很大,分析处理时涉及的数据范围也较广,往往涉及大规模的数据查询。提高系统性能,主要是提高系统的物理 I/O 性能,因为 I/O 瓶颈常成为影响系统性能的主要因素。在数据仓库设计中,应尽量减少每个查询要求的 I/O 处理次数,使每次 I/O 又能返回尽量多的记录。事实上,由于数据仓库的数据极少甚至不再更新,数据仓库的物理设计可以有更多的方法和途径来提高系统性能。下面介绍粒度划分和数据分割。

1) 粒度划分

对数据仓库开发者来说,划分粒度是设计过程中最重要的问题之一。所谓的粒度指数据仓库中数据单元的详细程度和级别。数据越详细,粒度越小级别就越低。数据综合度越高,粒度越大级别就越高。在传统的操作型系统中,对数据的处理和操作都是在详细数据级别上进行的,即采用最低级的粒度。但数据仓库环境中主要是分析型处理,粒度的划分将直接影响数据仓库中的数据量以及所适合的查询类型。一般需要将数据划分为:详细数据轻度总结、高度总结三级或更多级粒度。不同粒度级别的数据用于不同类型的分析处理。粒度的划分是数据仓库设计工作的一项重要内容,粒度划分适当与否是影响数据仓库性能的一个重要方面。

2) 数据分割

数据分割是数据仓库设计另一项重要内容,是提高数据仓库性能的一项重要技术。数据分割是指把逻辑上为统一整体的数据分割成较小的、可以独立管理的物理单元进行存储,便于重构、重组和恢复,以提高创建索引和顺序扫描的效率。数据的分割使数据仓库的开发人员和用户具有更大的灵活性。数据仓库中数据分割的概念与数据库中的数据分片概念是相近的。数据库系统中的数据分片有水平分片、垂直分片、混合分片和导出分片多种方式。水平分片是指按一定的条件将一个关系按行分为若干不相交的子集,每个子集为关系的一个片段。垂直分片是指将关系按列分为若干子集,垂直分片的片段必需能够重构原来的全局关系。

在进行数据仓库设计时,需要把数据分割与粒度划分结合起来考虑。

数据仓库中的元数据就是关于数据的数据,它描述了数据的结构、内容、码和索引等内容。传统数据库中的数据字典是一种元数据,但在数据仓库中,元数据的内容比数据库中的数据字典更丰富、更复杂。设计一个描述能力强、内容完善的元数据,是有效管理数据仓库,具有决定意义的重要前提。因此元数据的设计在整个数据仓库设计中占有重要的地位,是数据仓库设计的一个重要组成部分。

数据仓库中元数据的重要性表现在:

(1) 数据仓库服务于决策支持系统分析员以及高层决策人员,而这一部分人员往往把使用元数据作为分析的第一步。例如,数据仓库元数据中的广义索引中存有在每次数

据装载时产生的部分有关决策的数据。在做决策时,可以先查找这部分数据,再决定是否进行进一步的搜索。

(2) 操作型环境 and 数据仓库环境之间有着复杂的、多方面的区别。因此,从操作型环境到数据仓库的数据转换也是复杂的、多方面的。元数据应包含对这种转换的描述。元数据要将这种转换清晰地表示出来,把从哪些数据源用怎样的转换逻辑转换成数据仓库中的哪些目的数据等内容描述出来。这样,当从数据仓库向数据库回溯时,便能够根据数据变换的历史,找到原始依据。数据仓库的元数据还要将这种转换管理起来,既要保证这种转换是正确的、适当的和合理的,又要使其是可变的、灵活的。事实上,因为用户需求是不确定的,只有保证元数据的灵活性、可变性,才能真正保证其合理性和正确性。

(3) 除了描述和管理从数据库到数据仓库的转换外,数据仓库的元数据当然还要管理好数据仓库中的数据。一方面,数据仓库中的数据量很大,划分不同的粒度层次,进行分割策略的选择,以及建立各种各样的索引等,都需要在元数据中进行描述和管理。另一方面,数据仓库中包含着较长时期内的数据,不同时期的不同需求使得其数据从形式到内容都可能不同。

元数据的内容在数据仓库设计、开发、实施以及使用过程中不断完善,不仅为数据仓库的创建提供必要的信息、描述和定义,还为决策支持系统分析人员访问数据仓库提供直接的或辅助的信息。

14.3.3 数据转移技术

数据仓库的基本观念之一是,当数据从业务系统或其他数据源提取出来时,应该先经过变换或清洗,才能将它加载到数据仓库中。然而,对于数据转移的目的和实现转移的最优方法却存在很多混乱的看法。在数据仓库环境中进行数据转移的目的应该有两个:第一,改进数据仓库中数据的质量;第二,提高数据仓库中数据的可用性。

所谓数据转移也称数据转换或数据变换,就是把多种传统资源或外部资源信息中不完善的数据自动转换为商务中准确可靠的数据。为了便于讨论,我将把业务数据加载到数据仓库之前经历的内存和结构变化都纳入数据转移。

在建立数据仓库的过程中,数据转移是个重要的步骤。实施数据转移的方法很多,既有定制代码的程序,也有用于转移仓库数据的专门工具。无论采用什么方法,转移的几个基本方面是必须实现的。转移远远不止是在把数据移入数据仓库时改变它的数据结构,真正好的转移应能够在数据进入数据仓库时进行检验并提高它的质量和可用性。

为了对数据转移的复杂性进行深入的讨论,我们必须定义数据转移的几个基本类型。每一类都有自己的特点和表现形式。为了便于讨论,应该考虑以下四种转移类型:

(1) 简单转移。简单转移是所有数据转移的基本构成单元。数据处理一次只针对一

个字段,而不考虑相关字段的值。

(2) 清洗。清洗的目的是为了保证前后一致地格式化和使用某一字段或相关字段群。例如,可以包括地址信息的适当格式化。清洗还能检查某一特定字段的有效值,通常是进行范围检查或从枚举清单中做出选择。

(3) 集成。集成是指将业务数据从一个或几个来源中取出,并逐字段地将数据映射到数据仓库的新数据结构上。

(4) 聚集和概括。聚集和概括是把业务环境中找到的零星数据压缩成数据仓库环境中的较少数据块。有时对细节数据进行聚集是为了避免数据仓库存入业务环境中那样具体的数据,有时则是为了建立包括数据仓库聚集副本或概括副本的数据商场。

顾名思义,简单转移是数据转移中最简单的形式。这种转移一次只改变一个数据属性而不考虑该属性的背景或与其它相关的其他信息。简单转移有如下形式:

(1) 数据类型转换。最常见的简单变换是转换一个数据元的类型。除了将一个空值改为空白或零之外(或反之),无须改变该数据元的语义值就可以完成一次简单的数据类型转换。当应用程序存储的某个类型的数据只在该应用程序的背景下才有意义,在企业水平上却没有意义时,就常常要求进行这类变换。这类转换可以通过编码程序中的简单程序完成,或者运用数据仓库的数据转换工具完成。所有转换工具都能轻松、迅速地这类简单转换。另外,许多简单变换可以通过数据库卸载或加载设施完成,这种设施可以在从平面文件中取出数据并加载到数据库中时进行简单的数据类型转换。

(2) 日期/时间格式的转换。一般而言,在设计和构造业务应用程序时,对各程序内的日期/时间很少进行统一的处理。应用程序设计人员常常选用最适合的包含业务数据的数据库管理系统的格式表示日期和时间。在某些情况下,甚至在同一应用程序内日期和时间的处理也不一致。程序模块的设计者按照他们认为合适的方式随意设计日期和时间字段,很少考虑到一致性。在处理业务系统中时间和日期的格式差异的背后,无论有什么原因,有一件事是清楚的:数据仓库必须用单一的模式识别日期和时间信息。正如2000年问题所显示的,我们必须用稳健的方式存入日期,包括完整的年份。选择日期和时间的实际物理表示法决定于许多因素,但无论选择哪种表示方式,都应该存入年份的四个数字,而且在整个数据仓库中必须一致。

(3) 字段解码。最后一类简单变换是编码字段的解码。简单地说,数据一般不应该以编码的格式存放在数据仓库中。我们在业务数据库中建立代码是为了节省数据库存储空间。虽然人不理解这些代码,但这并不是大问题,因为我们与那些代码的交互是由应用程序管理的,这些程序在必要的时候会为我们破解那些值的代码。在数据仓库环境中,情况就大不一样了。因为有些查询工具能破解数据库值,所以无法保证应用程序逻辑一定能将用户与数据仓库数据库中的编码值隔绝开。考虑到查询工具的不断增长以及大多数

数据仓库用户自己编写查询时,情况确实如此。当他们编写或运行的特别查询程序,用户看到的信息就像存储在数据库中一样。因为用户可能来自公司的任何部门,所以数据仓库的所有用户不可能都有足够的背景知识和必要的培训,使他们能够理解业务数据库中使用的编码值。因此,业务系统和外部数据中的编码值在存入数据仓库之前,应该转换为经过解码且易于理解的相应值。当然,这样做必须遵循正确的路线。一方面,我们想把编码值充分扩展,使它们为大多数用户理解。另一方面,把一个值扩展得太多要占用额外的存储空间,而且把该值当作查询中的检索标准也很困难。还必须考虑到用户对于数据元业务含义的熟悉程度。从技术角度看,字段解码是一个非常易于实现的过程,可以很容易地结合到转移程序中去,也可以在数据转移工具中轻松地完成。然而,确定应该进行多少解码工作是很困难的。

清洗是比简单变换更复杂的一种数据转移。在这种转移中,要检查的是字段或字段组的实际内容而不仅是存储格式。

一种清洗是检查数据字段中的有效值。这可以通过范围检验、枚举清单和相关检验来完成。这里给出了针对每种方法的例子。

范围检验是数据清洗最简单的形式。它检验每个字段中的数据以保证它位于预期范围之内,通常是数字范围或日期范围。例如,可以检验一个发票编号,看它是否是有效的发票编号,即编号是否介于 1000 和 99999 之间。也可以检验发票日期,看它是否位于 1995 年 4 月 1 日(公司开业之日)和当前日期之间,任何位于该日期范围之外的发票都应该剔除出去。

枚举清单也相对容易实现。这种方法是对照数据字段可接受值的清单检验该字段的值。例如,可以检验一下送货类型代码,看看它是否属于有效值“快递”或“普通”之中的一个,任何与这张清单不符的值将被剔除出去以待进一步调查。

相关检验稍微复杂一些,因为它要求将一个字段中的值与另一个字段中的值进行对比。例如,我们可以检验发票记录上的采购订货编号字段,看它是不是我们系统中存在的采购订货。任何含有采购订货编号却没有相应采购订货的发票记录应该留待进一步调查。

数据清洗的另一主要类型是重新格式化某些类型的数据。这种方法适用于以许多不同方式存储在不同数据来源中的信息,必须在数据仓库中把这类信息转换成一种统一的表示方式。最需要格式化的信息之一是地址信息。由于没有一种获取地址的标准方式,所以同一个地址可能有许多不同表示方式。当然,当人读这些地址时,他们能认出它们指的是同一地点。但在数据仓库中,如果地址按不同方式存储,那种对应关系就丢失了。因此我们必须把地址转变成一种共同的格式,这样我们才能在数据仓库中确定哪些地址其实是相同的。

集成是要把从全然不同的数据源中得到的业务数据结合在一起,困难在于将它们集成为一个紧密结合的数据模型。这是因为数据必须从多个数据源中提取出来,并结合成为一个新的实体。这些数据来源往往遵守的不是同一套业务规则,在生成新数据时,必须考虑到这一差异。因为不可能识别出每种可能发生的集成,所以这里只讨论字段水平的简单映射。

字段水平的简单映射在必须执行的数据转移总量中占了大部分。在一个典型的数据仓库里,指定进行的转移中有 80%~90% 是字段水平的简单映射。这种映射是指数据中的一个字段被转移到目标数据字段中的过程。例如,从一个业务数据库中得到一条记录,把它的一个字段转移到数据仓库的数据结构中。在此过程中,这个字段可以利用前面讨论过的任何一种简单变换进行变换。这些字段水平的映射很容易实现,并且占了数据集成工作的大部分。

大多数数据仓库都要用到数据的某种聚集和概括。这通常有助于将某一实体的实例数目减少到易于驾驭的水平,也有助于预先计算出广泛应用的统计概括数字,以使每个查询不必计算它们。虽然聚集和概括往往被当作可以互换的名词来使用,但在数据仓库中,这两个名词的含义还是有一些差异的。概括是指按照一个或几个业务维将相近的数值加在一起,例如商店把每日的销售额加在一起,生成按地区计算的月销售额。聚集指将不同业务元素加在一起成为一个公共总数。但为了便于讨论,我们不区分聚集和概括,在数据仓库中它们是以相同的方式进行的。

有时,数据仓库中存放的具体数据的具体程度往往不如业务系统中存放的细节数据。这时,就有必要在变换业务数据的过程中加入一些数据聚集功能。这可以减少存储在数据仓库中的行数。有时,聚集也可以用于建立数据商场,这类数据商场是从仓库中存储的更具体的数据中衍生出来的。聚集的第三个作用是去除数据仓库中过时的详细数据。在许多情况下,数据在一定时期内要以很具体的详细数据存放着。一旦数据到达某一时限,对这些详细数据的需求就大大减弱了。此时,这些非常具体的数据应该传送到备用存储器中,而数据的概括形式则可以存放在数据仓库当中。

当然,对于各种细节数据有许多概括方法,每一种概括或聚集都可以在某一个或几个维进行。例如,一家零售组织保存着每次单个销售事务的事务处理数据。以这一细节数据为基础,可以从许多维进行概括。例如产品大类、销售事务类型、地理区域、顾客和时间期限。生成的每一个总和都是在一个或几个维上进行聚集的结果。这样,用户通过访问元数据,就能够得知每种概括数据的衍生方法。这一点是极为重要的,只有这样才能知道哪些维已经得到了概括以及概括的程度。

目前可以得到的数据清洗工具许多都已内置了概括功能,尤其是在时间维上进行聚集的功能。当然,通过使用分类设施就可以轻松地做到这一点。这类设施中具有复杂的

概括逻辑能力。不管究竟是如何做到这一点的,重要的是用户能够轻松地访问元数据,了解生成总和数据所用的标准。

在数据仓库环境中,实现数据转移的方法当然不止一种。但最主要的是选择转移方式——使用为数据仓库提供数据的专用数据转移工具,还是编制能实现转换逻辑的程序。哪一个才是最佳选择决定于很多因素:

(1) 时间范围。虽然使用数据转移工具能大大方便建立和维护数据仓库的过程,但获取、配置和学习这些工具都要花时间。如果交付第一个数据仓库产品的时间限制很紧,那么这样的项目往往需要选择人工编码。在数据仓库建设得较完善,项目小组的可信度较高时再迁移到数据转移工具中。

(2) 预算。数据仓库工具可能会很贵,但编程人员编写变换程序的时间也一样宝贵,因此最佳选择取决于哪类预算近期更容易得到。但从长远的观点来看,采用现成的工具比手工编制和维护变换程序的成本可能要节省得多。

(3) 数据仓库的规模。范围很小的数据仓库(即数据源很少,要实现的转移也很少)可能无法证实使用数据转移工具的成本的合理性。然而,当数据仓库的范围逐渐扩大时,维护手工编写的变换程序将变得越来越困难,工具就会发挥更大的作用。但对于一个规模很小的初始数据仓库来说,当然就不需要变换工具了。

(4) 数据仓库项目小组的规模和技能。如果数据仓库小组足够大,而且具有适当的编程技巧,建立和维护转移逻辑就容易一些。但是,大多数数据仓库小组都受到资源的限制,无法得到足够的开发时间以维护所有的变换代码。因此他们发现转移工具特别有用,因为数据仓库分析人员也许不需要编程人员的帮助就可以自己建立并维护大多数的数据转移。

当然,使用数据转移工具最主要的原因之一与节省时间和有效利用成本没有太大的关系,因为好的数据转移工具能自动地生成并维护宝贵的元数据。

14.3.4 OLAP 技术

20 世纪 60 年代末,E. F. Codd 所提出的关系数据库模型促进了关系数据库及联机事务处理(On-Line Transaction Processing, OLTP)的发展。数据不再以文件方式同应用程序捆绑在一起,而是分离出来以关系表方式供大家共享。随着政府及商业应用的发展,数据量越来越大,同时用户的查询需求也越来越复杂,涉及的已不仅是查询或操纵一张关系表中的一条或几条记录,而是要对多张表中的千万条记录进行数据分析和信息综合。关系数据库系统已不能全部满足这一要求。操作型应用和分析型应用在性能上尤其难以两全,尽管为了提高性能,人们常常在关系数据库中放宽了对冗余的限制,引入了统计及综合数据。但这些统计及综合数据的应用逻辑却是分散杂乱的,非系统化的,因此分析功能

有限,不灵活,维护困难。在国外,不少软件厂商采取了发展其前端产品来弥补关系数据库管理系统支持的不足,他们通过专门的数据综合引擎,辅之以更加直观的数据访问界面,力图统一分散的公共应用逻辑,在短时间内响应非数据处理专业人员的复杂查询要求。1993年,E. F. Codd 将这类技术定义为联机分析处理(On-Line Analytical Processing,OLAP)。OLAP 作为一类产品同 OLTP 明显区分开来。

OLAP 是针对特定问题的联机数据访问和分析。为了反映用户所能理解的企业的真实的“维”,原始的数据被进行了转换,从而形成了可用的信息。通过多种可能的观察形式对信息进行快速和稳定一致的交互性存取,允许管理决策人员对数据进行深入观察。

OLAP 是以数据仓库作为分析决策的基础,针对特定问题的联机数据访问和分析,OLAP 能够基于某个或多个角度对不同数据集合进行比较,能够从不同角度切割数据集合从而进行分析。从某种意义上说,OLAP 是有预见性的。OLAP 的分析建立在经验的基础上,对数据进行某种指定关联的分析。在联机事务处理系统中,由于数据的离散性,使 OLAP 实现起来相当复杂甚至不可能。而以数据仓库为依托,辅之以 OLAP 工具,OLAP 的实现将十分简单易行。

OLAP 中的基本概念如下所示。

(1) 变量:变量是数据的实际意义,即描述数据“是什么”。例如数据“10000”本身并没有意义或者说意义未定,它可能是一个学校的学生人数,也可能是某产品的单价,还可能是某商品的销售量等。一般情况下,变量总是一个数值量度指标,例如:“人数”、“单价”和“销售量”等都是变量,而“10000”则是变量的一个值。

(2) 维:维是人们观察数据的特定角度。例如,企业常常关心产品销售数据随着时间推移而产生的变化情况。这是从时间的角度来观察产品的销售,所以时间就是一个维,简称为时间维。企业也时常关心自己的产品在不同地区的销售分布情况,这是从地理分布的角度来观察产品的销售,所以地理分布也是一个维,称为地理维。

(3) 维的层次:人们观察数据的某个特定角度(即某个维)还可以存在细节程度不同的多个描述方面,我们称这多个描述方面为维的层次。一个维往往具有多个层次,如描述时间的维,可以从日期、月份、季度和年等不同层次来描述,那么日期、月份、季度和年等就是时间维的层次;同样,城市、地区和国家等构成了一个地理维的多个层次。

(4) 维成员:维的一个取值称为该维的一个维成员。如果一个维是多层次的,那么该维的维成员是在不同维层次上的取值的组合。例如,我们考虑时间维具有日期、月份和年这三个层次,分别在日期、月份和年上各取一个值组合起来,就得到了时间维的一个维成员,即“某年某月某日”。一个维成员并不一定在每个维层次上都要取值,例如,“某年某月”、“某月某日”和“某年”等都是时间维的维成员。对应一个数据项来说,维成员是该数据项在某维中位置的描述。如对一个销售数据来说,时间维的维成员“某年某月某日”就

表示该销售数据是“某年某月某日”的销售数据,“某年某月某日”是该销售数据在时间维上的位置描述。

(5) 多维数组: 一个多维数组可以表示为(维 1, 维 2, ..., 维 n , 变量)。例如,(地区, 时间, 销售渠道, 销售额)就是一个多维数组,其中销售额是变量,它定义在地区维、时间维和销售渠道维这三者的基础上。

(6) 数据单元: 多维数组的取值称为数据单元。当在多维数组的各个维中都选中一个维成员时,这些维成员的组合就惟一地确定了一个变量的值。那么数据单元就可以表示为(维 1 的维成员, 维 2 的维成员, ..., 维 n 的维成员, 变量的值)。

多维分析是指对以多维形式组织起来的数据采取切片、切块和旋转等各种分析动作,以求剖析数据,使最终用户能从多个角度,多侧面地观察数据库中的数据,从而深入地了解包含在数据中的信息和内涵。多维分析方式迎合了人的思维模式,因此减少了混淆,并且降低了出现错误解释的可能性。多维分析的基本动作如下。

(1) 切片: 在多维数组的某一维上选定一维成员的动作称为切片,即在多维数组(维 1, 维 2, ..., 维 n , 变量)中选一维,并取其一维成员。

(2) 切块: 在多维数组的某一维上选定某一区间的维成员的动作称为切块,即限制多维数组的某一维的取值区间。显然,当这一区间只取一个维成员时,即得到一个切片。

(3) 旋转: 旋转就是改变一个报告或页面显示的维方向。例如,旋转可能包含了交换行和列,把某一个行维移到列维中去,或是把页面显示中的一个维和页面外的维进行交换(令其成为新的行或列中的一个)。

OLAP 的数据来源于数据库。通过 OLAP 服务器,将这些数据抽取和转换为多维数据结构,以反映用户能理解的企业的真实维。通过多维分析工具以多个角度、多个侧面对信息进行快速、一致和交互的存取,从而使分析员、经理和行政人员能够对数据进行深入地分析和观察。

在数据仓库系统中,OLAP 使用的多维数据可以位于不同的层次,可以作为数据仓库的一部分,也可以作为数据仓库工具层的一部分。由于所处层次的不同,其分析结果的综合程度也相应有高低之分,所以可以满足具有不同应用需求用户的要求。

1993 年,E. F. Codd 提出了有关 OLAP 的 12 条准则,这也是他继关系数据库和分布式数据库的两个“12 条准则”后提出的第三个“12 条准则”。尽管业界对这个 12 条准则褒贬不一,但其主要方面,如多维数据分析、客户机/服务器结构、多用户支持及一致的报表性能等得到了大多数人的认可。

(1) OLAP 模型必须提供多维概念视图: 从用户分析员的角度来看,整个企业的视图在本质上是多维的,因此 OLAP 的概念模型也应是多维的。企业决策分析的目的不同,决定了分析和衡量企业的数据总是从不同的角度来进行的,所以企业数据空间本身就

是多维的。

(2) 透明性准则：无论 OLAP 是否是前端产品的一部分，对用户来说，它都是透明的。如果在客户机/服务器结构中提供 OLAP 产品，那么对最终分析员来说，它同样也应透明。透明性原则包括两层含义：首先，OLAP 在体系结构中的位置对用户是透明的。OLAP 应处于一个真正的开放系统结构中，允许分析工具嵌入到分析人员指定的任何位置而不影响嵌入工具的性能，这对保持用户现有的效率，保证良好的性能至关重要。同时必须保证 OLAP 的嵌入不会引入和增加任何复杂性。其次，OLAP 的数据源对用户也是透明的。用户只需使用熟悉的查询工具进行查询，而不必关心输入 OLAP 工具的数据来自同构还是异构的企业数据源。

(3) 存取能力准则：OLAP 系统不仅能进行开放的存取，而且还提供高效的存取策略。OLAP 用户分析员不仅能在公共概念视图的基础上对关系数据库中的企业数据进行分析，而且在公共分析模型的基础上还可以对关系数据库、非关系数据库和外部存储的数据进行分析。OLAP 系统应提供高效的存取策略，应使系统只存取与指定分析有关的数据，避免多余的数据存取。

(4) 稳定的报表性能：当数据的维数和综合层次增加时，提供给最终分析员的报表能力和响应速度不应该有明显的降低和减慢，这对维护 OLAP 产品的易用性和低复杂性至关重要。即便企业模型改变时，关键数据的计算方法也无须更改。只有做到这一点，OLAP 工具提供的数据报表和所做的预测分析结果才是可信的。

(5) 客户机/服务器体系结构：OLAP 是建立在客户机/服务器体系结构上的。这要求它的多维数据库服务器能够被不同的应用和工具访问到。服务器端能够以最小的代价智能地完成同多种服务器之间的挂接任务。服务器端必须完成分散的企业数据库的逻辑模式和物理模式之间的映射，并确保它们的一致性，从而保证透明性和建立统一的公共概念模式、逻辑模式和物理模式。客户端负责应用逻辑和用户界面。

(6) 维的等同性准则：每一数据维在数据结构和操作能力上都是等同的。系统可以将附加的操作能力赋予所选的维，但必须保证该操作能力可以赋予其他任意的维，即要求维上的操作是公共的。

(7) 动态的稀疏矩阵处理准则：OLAP 工具的物理模型必须充分适应指定的分析模型，提供“最优”的稀疏矩阵处理，这是 OLAP 工具所应遵循的最重要的准则之一。该准则包括两层含义：第一，对任意给定的稀疏矩阵，存在且仅存在一个最优的物理视图，它能提供最大的内存效率和矩阵处理能力。稀疏度是数据分布的一个特征，如果不能适应数据集合的数据分布，将会导致快速、高效操作的失败。第二，OLAP 工具的基本物理数据单元能够配置给可能出现的维的子集。同时，还要提供动态可变的访问方法并包含多种存取机制，使得访问速度不会因数据维的多少和数据集的大小而变化。

(8) 多用户支持能力准则：多个分析员可以同时工作于同一分析模型上，或者可以在同一企业数据上建立不同的分析模型。该准则可由准则 5 推出。OLAP 工具必须提供并发访问、数据完整性及安全性机制。实际上，OLAP 工具必须支持多用户也是为了适应数据分析工作的特点。我们推荐以工作组的形式来使用 OLAP 工具，这样多个用户可以交换各自的想法和分析结果。

(9) 不受限的跨维操作：多维数据之间存在固有的层次关系，这就要求 OLAP 工具能自己推导出而不是由最终用户明确定义相关的计算。对于无法从固有关系中得出的计算，要求系统提供计算完备的语言来定义各类计算公式。该准则是对准则 1 的补充，对操作能力和操作范围做出了要求。

(10) 直观的数据操纵：要求数据操纵直观易懂。综合路径重定位、向上综合、向下挖掘和其他操作都可以通过直观方便的点拉操作完成。

(11) 灵活的报表生成：报表必须从各种可能的方面显示从数据模型中综合出的数据和信息，充分反映数据分析模型的多维特征。

(12) 不受限制的维数与聚集层次：OLAP 工具的维数应不小于 15 维，用户分析员可以在任意给定的综合路径上建立任意多个聚集层次。

14.3.5 企业决策支持解决方案

这一节介绍企业决策支持解决方案，我们将从决策支持系统的体系结构、软件工具和开发 3 个方面进行讨论。

在从事任何重要决策支持系统开发之前，开发者都应该对它的体系结构有一个清楚的概念。这有助于开发者将对决策支持系统的看法与管理层进行沟通，进而帮助开发者规划出在未来也能继续满足用户需求的系统。

计算机的应用程序在一个包括硬件和操作系统的环境中运行，决策支持系统也不例外。如果涉及到规划决策支持系统，用户将需要选择其运行环境。总的来说，就是定义它的体系结构。在该体系结构中，用户可以在大型主机、小型计算机和微型计算机中做出选择。另外，还要做出其他选择。不同用户所做的选择其区别在于硬件与使用的操作系统。用户还必须考虑连接若干台计算机并分担工作负荷的不同途径。每种选择都有其优点和缺点。选择最佳的组合能帮助用户决策支持系统更快地响应和更加成功，而选择不当将导致失败。

决策支持系统的体系结构需要实现以下目标。

- (1) 系统的互操作性，使信息能快捷地传送。
- (2) 系统的兼容性，使资源能在整个组织范围内方便而有效地共享。
- (3) 系统的可扩展性，使单一功能的组成部分不会成为阻碍组织发展的瓶颈。

决策支持系统的体系结构必须能反映下列元素。

(1) 决策支持系统的数据库,包含任何已存在的数据库、内部数据库和外部数据库(对于组织而言),以及任何专门为决策支持系统所建立的数据库。体系结构应说明,谁对于个人、部门和企业层次上的不同数据库类型负责,保证它们的准确性、并发性和安全性。

(2) 决策支持系统的模型,包括数据源的信息、维护这些资源的组织责任以及存取数据源的限制等。在初始阶段,说明每个独立模型也许是不实际的,但定义主要的类别是可能的。

(3) 决策支持系统的用户,有关他们的地点、工作和教育水平的假设,以及各种可能影响他们使用决策支持系统的因素。

(4) 软件工具,通过它们,用户可访问数据库和模型,系统管理员可管理数据库和模式。

(5) 硬件和操作系统平台,在这一层次中有数据库和模式,程序也运行在这一层,用户通过这一层访问决策支持系统。

(6) 网络和通信功能,通过这些功能平台可互相连接,这些必须能够反映个人与一个或多个服务器或数据库连接的需要,反映工作组织之间相互通信的需要,反映企业与每个工作之间的相互连接和共享数据。

客户机/服务器模式是当今比较常见的计算模式。在决策支持系统中,服务器系统作为数据仓库,客户机系统通常在用户的台式计算机上运行应用程序,并使用来自服务器的数据。使服务器和客户机的能力都非常适应它们各自在整个系统中承担的角色。

采用客户机/服务器模式的优点如下所述。

(1) 客户机/服务器模式允许综合应用程序的每一部分均运行在为该部分单独优化的硬件/软件环境中,可以缩小系统的整体成本。

(2) 系统的模块化性质简化了日后扩充的需求。客户机可以添加到网络中或独立于服务器升级,也可以不影响客户机只升级服务器。

(3) 客户机/服务器模式比较流行,许多应用程序和软件开发工具均设计成在客户机/服务器环境下运行。使用这些应用程序和软件开发工具,可以降低开发时间和成本。

但客户机/服务器模式也有缺点。

(1) 在客户机/服务器系统中,应用程序开发和管理的复杂性大。

(2) 安全成为主要内容。任何时候,用户均通过网络访问关键的公司数据。系统开发者必须能敏感地应付未预见访问的可能性。安全特性也许涉及限制组织外部访问的防火墙,它是复杂的附加层。这些特性在某些方面增加了整体开支,例如,安全软件的费用,安装和管理该软件人员的费用,附加的硬件资源费用。

决策支持系统中一般均用到数据仓库技术。只有为了达到优化数据仓库的目的,才

可能更新决策支持系统的硬件,从而减少硬件开支。决策支持系统的硬件不需与其他应用程序共享,因此无论在组织中其他地方发生什么事情,决策支持系统响应时间均不会变坏。单一数据仓库的环境使得决策支持系统更简单并且不易出错。

决策支持系统是一种信息系统,可以按照与其他任何信息系统同样的构造方法来建造。通常有 4 种获取软件功能的基本方法:购买集成的软件包;定制软件包;使用专用的工具软件;使用编程语言编写所要的程序。既然决策支持系统是一种软件,这些途径同样适用于它。选择定制或非定制软件包,还是选择开发软件,通常由以下两方面因素决定:

(1) 用户的需要与组织内其他大多数需要的相似程度。这种相似程度决定了用户是否可以在市场上以较低的花费来获取满足系统要求的软件包,避免花更大的代价来开发软件。

(2) 应用程序的价值影响力。这决定了准确地得到用户需要功能的价值与用户在标准软件包中能够得到的功能之比。组织的规模是影响因素之一。如果通用汽车公司投资 30 万美元建立一个系统提高了销售总额百分之一的利润,这一投资就是明智的。如果生产三明治和汽水的小公司同样投资 30 万美元,结果也提高了销售总额的百分之一,那么这样的投资就不太明智。

用于辅助决策支持系统开发的专用软件主要有以下几类:

- (1) 数据库管理软件包;
- (2) 信息检索软件包;
- (3) 专用建模软件包和语言;
- (4) 数据统计分析软件包;
- (5) 预测软件包;
- (6) 图形软件包。

决策支持系统可以用不同的方法进行构造,典型的方法是生命周期法和原型法。

系统开发生命周期法(system development life-cycle, SDLC)常用于开发管理信息系统等比较结构化的系统。但是,如何将生命周期法应用于决策支持系统?如何设计决策支持系统以改进管理者所面临的半结构化或非结构化问题的决策过程?事实上,开发者不能完全了解用户的需求,在大多数决策支持系统的分析设计开始阶段,用户并不确切知道自己的需求。因此,必须重视在设计过程中不断学习,也就是说,作为设计和实现的一部分,希望用户更多地了解问题或环境,以识别新的未预料的信息需求。

原型法用下列方式构造决策支持系统,即经过一系列短时间的开发步骤,在这些步骤中有来自用户的中间反馈,以保证开发正确进行。因此,决策支持系统工具必须能适应快速和容易变化的需求。迭代过程包括下列四项任务。

- 1) 首先构造/选择重要的子问题

用户和构造者一起识别一个重要的子问题,用于初始决策支持系统的构造。这项开发初期的联合工作可以在项目参与者之间建立初步的工作关系,并建立相互沟通的渠道。子问题应该足够简明,对于问题的本质、计算机支持的需求及特性都是清楚的,并且决策人对该问题有很大的兴趣,哪怕决策人的兴趣是短暂的。

2) 为决策人开发一个小的可用系统

这里并不进行详细的系统分析或可行性分析,虽然是小规模的开发,构造者和用户实际上仍然快速地经历了系统开发的所有步骤。除了必要的部分以外,系统应当是简单的。

3) 不断地进化系统

在各循环结束时,由用户和构造者评价系统。评价是开发过程的一部分,并且是整个迭代设计过程的控制机制。评价机制保证决策支持系统的开发工作和费用与其价值相一致。在评价结束时,决定是否进一步细化决策支持系统,或者停止。

4) 不断地细化、扩展和修改系统

不断循环地扩展和改进决策支持系统版本,所有分析、设计、构造、实现和评价步骤在各个细化过程中重复进行。

该过程经多次重复、不断进化直到产生一个比较稳定和综合的系统。在这个过程中,用户、构造者和技术人员之间的交流是极为重要的。用户和构造者一起合作,用户在使用和评价中起主要作用。而构造者在设计和实现阶段中起主要作用。这里,用户起主动作用,这与常规系统的开发是不同的。在常规系统开发过程中,用户常常是被动的。需要注意的是,数据的说明随系统的进化而逐步产生。

这种迭代设计方法产生一个特定的决策支持系统,对于为个人提供支持的决策支持系统设计,该过程是比较直接的,对于为群体或组织提供支持的决策支持系统,该过程会变得更复杂,需要特别为用户和构造者建立相互沟通的机制。当维护适用于所有用户的标准核心系统时,还需要提供对于个人变化的支持机制。

迭代过程概括如下:它开始于部分问题的模型或整个问题的简化情况。这给最终用户某些具体的概念。然后,最终用户提供可能改进决策支持系统的建议。接着,开发出决策支持系统的新版本,该过程继续下去直到最终用户对系统满意为止。由于在复杂的决策过程中,用户常不能精确地知道他们需要什么,并且决策支持系统构造者也不了解最终用户需要或接受什么,而迭代过程使他们可以相互学习,故该迭代过程是必需和有效的。

原型法有下列主要优点:

- (1) 开发时间短;
- (2) 用户反馈速度快;
- (3) 用户对系统及其信息需求和功能的理解大大增强;
- (4) 费用低。

当使用该方法时,可能会失去在系统生命周期法各阶段中可得到的东西,如对系统效益和费用的整体了解、企业信息需求的详细描述、容易维护和较好测试的信息系统以及已有较好准备的用户。

管理者对信息需求的认识可能不清楚,所以大多数决策支持系统都用原型法进行开发。开发的过程大致如下。

① 规划: 主要涉及评价和问题诊断。要定义决策支持的目的和目标。规划的关键是确定由决策支持系统支持的关键决策,如证券选择系统,关键的决策是根据特定顾客的要求选择正确的股票。

② 调查: 确定用户需要和可用的资源,如硬件、软件、经销商的相关经验和相关研究综述,还需要仔细分析决策支持系统的环境。

③ 系统分析和概念设计: 确定最好的开发方法和系统实现所需要的资源,包括技术、财务和组织的资源。在概念设计完成后,进行可行性分析。

④ 设计: 确定系统部件、结构和特点的详细说明。对应决策支持系统各主要部件,设计可分为四个主要部分: 即数据库及其管理系统、模型库及其管理系统、知识库及其管理系统(如果需要)以及人机界面。需要选择合适的软件或编写程序。

⑤ 构造: 根据设计的原理和使用的工具,决策支持系统的构造可以有不同的方式。构造是设计方案的技术实现,应不断测试和改进。

⑥ 实现: 实现包括下列任务,其中有些任务可以同时进行。

- 测试。收集系统输出数据,并与设计说明进行比较。
- 评价。评价已实现的系统对用户需求的满足程度。因为系统在不断地修改或扩展,所以没有确切定义的完成日期或用于比较的标准,且测试和评价通常会引起设计和构造的变化,整个过程需周期性地反复几次。因此,对决策支持系统的评价是比较困难的。
- 演示。为用户演示完整的系统功能是一个重要阶段,这会使用户较容易接受系统。
- 适应。为用户提供掌握系统基本功能和操作的说明。
- 训练。按系统的结构和功能,训练用户上机操作,并训练用户如何维护系统。
- 配置。配置完整的运行系统,供所有的用户使用。

⑦ 文档与维护: 维护包括为系统及其用户提供支持的计划,并开发系统使用和维护的文档。

⑧ 适应: 适应用户日常需求以及今后的变化。如有必要,可重复以上过程并改进系统。

14.3.6 联机事务处理

在这一节中,我们将联机分析处理和联机事务处理(OLTP)进行分析和比较。

联机分析处理(OLAP)主要是关于如何理解聚集的大量不同的数据。与联机事务处理应用程序不同,联机分析处理包含许多具有复杂关系的数据项。联机分析处理的目的就是分析这些数据,寻找模式、趋势以及例外情况。

联机分析处理是决策人员和高层管理人员对数据仓库进行信息分析处理。联机分析处理数据可能包含以地区、类型或渠道分类的销售数据。一个典型的 OLAP 查询可能要访问一个多年的销售数据库,以便能找到每一个地区每一种产品的销售情况。当得到这些数据后,分析人员可能会进一步地细化查询,在以地区和产品分类的情况下查询每一个销售渠道的销售量。最后,分析人员可能会针对每一个销售渠道进行年与年或者季度与季度的比较。整个过程必须被联机执行并要有快速的响应时间,以便分析过程不受外界干扰。联机分析处理可以被刻画为具有下面特征的联机事务。

(1) 可以存取大量的数据,比如几年的销售数据,分析各个商业元素类型之间的关系,如销售、产品、地区和渠道。

(2) 需要包含聚集的数据,例如销售量、预算金额以及消费金额。

(3) 按层次对比不同时间周期的聚集数据,如月、季度或者年。

(4) 以不同的方式来表现数据,如以地区、每一地区内按不同销售渠道或者不同产品来表现。

(5) 需要包含数据元素之间的复杂计算,如在某一地区的每一销售渠道的期望利润与销售收入之间的分析。

(6) 能够快速地对应用户的查询,以便用户的分析思考过程不受系统影响。

OLAP 服务器允许用熟悉的工具方便地存取不同的数据源。快速响应时间是 OLAP 中的关键因素。它分批处理报表,应用程序中的信息必须快速可得,以便执行进一步的分析。为了使分析过程变得容易,OLAP 应用程序经常以诸如电子表格这样容易辨识的形式提交数据。

在 OLTP 中,操作人员和低层管理人员利用计算机网络对数据库中的数据进行查询、增加、删除和修改等操作,以完成事务处理工作。

OLTP 以快速事务响应和频繁的数据修改为特征;用户利用数据库快速地处理具体业务。OLTP 应用有频繁的写操作,所以数据库要提供数据锁和事务日志等机制。OLTP 应用要求多个查询并行,以便将每个查询的执行分布到一个处理器上。

与 OLAP 应用程序不同,OLTP 应用程序包含大量相对简单的事务。这些事务通常只是需要获取或更新其中的一小部分数据,且这些表之间的关系通常是很简单的。

现代的数据库存储有数以万计的数据,每天经常处理成千上万的事务。OLTP数据库在查找业务数据时是非常有效的,但在为决策者提供综合汇总性数据时则显得力不从心。这就需要 OLAP 技术。OLAP 是一项以灵活、可用和及时的方式构造、处理和表示综合数据的技术。例如,假设有下面一个简单的问题:查看 1999 年西南地区的销售情况,数据按省、季度和产品分类。首先要从 OLTP 数据库中抽取数据,这需要大量的时间。然后,还要用大量的时间来查询该年四个季度每个月的销售数据等。而用 OLAP 技术则可以在几秒钟内完成这样的问题。

OLTP 的特点在于事务量大,但事务内容比较简单且重复率高。大量的数据操作主要涉及一些增加、删除和修改操作,但一般仅仅涉及一张或几张表的少数记录。因此 OLTP 适合于处理高度结构化的信息。与其相适应,在数据组织方面 OLTP 以应用为核心,是应用驱动的,数据模型采用 E-R 模型。

下面将 OLAP 与 OLTP 加以对比。

OLTP 处理的数据是高度结构化的,涉及的事务比较简单,因此复杂的表关联不会严重影响性能。反之,决策支持系统的一个查询可能涉及数万条记录。这时复杂的连接操作会严重影响性能。在 OLTP 系统中,数据访问路径是已知的,至少是相对固定的,应用程序可以在事务中使用具体的数据结构如表和索引等。而决策支持系统使用的数据不仅有结构化数据,而且有非结构化数据,用户常常在想要某种数据前才决定分析该数据。因此数据仓库系统中一定要为用户设计出更为简明的数据分析模型,这样才能为决策支持提供更为透明的数据访问。

OLAP 是以数据仓库为基础的,其最终数据来源与 OLTP 一样均来自底层的数据库系统,但由于二者面对的用户不同——OLTP 面对的是操作人员和低层管理人员,OLTP 面对的是决策人员和高层管理人员,因而数据需求特点与数据处理方式也明显不同。OLTP 和 OLAP 的区别如表 14-1 所示。

表 14-1 OLTP 与 OLAP 对比表

OLTP	OLAP
数据库原始数据	数据库导出数据或数据仓库数据
细节性数据	综合性数据
当前数据	历史数据
经常更新	不可更新,但周期性刷新
一次性处理的数据量小	一次性处理的数据量大
响应时间要求高	响应时间合理
用户数量大	用户数量相对较少
面向操作人员,支持日常操作	面向决策人员,支持管理需要
面向应用,事务驱动	面向分析,分析驱动

由表 14-1 可见,OLTP 与 OLAP 是两类不同的应用。OLTP 面对的是操作人员和基层管理人员,OLAP 面对的则是决策人员和高层管理人员。OLTP 是对基本数据的查询、增加、删除以及修改等操作处理,以数据库为基础。而 OLAP 更适合以数据仓库为基础的数据分析处理。OLAP 所需的历史的、导出的及经综合提炼的数据均来自 OLTP 所依赖的底层数据库。相比 OLTP 数据而言,OLAP 数据要增加数据多维化或预综合处理等操作。例如,对一些统计数据,首先要进行预综合处理,建立不同层次级别的统计数据,从而满足快速统计分析和查询的要求。除了数据及处理上的不同之外,OLAP 前端产品的界面风格及数据访问方式也同 OLTP 有所区别。OLTP 多为处理操作人员经常用到的固定表格,查询和数据显示也比较固定和规范。而 OLAP 多采用便于非数据处理专业人员理解的方式,如多维报表和统计图形等,查询及数据输出直观灵活,用户可以方便地进行逐层细化、切片、切块和数据旋转等操作。

第 15 章 知识产权基础知识

15.1 知识产权的概念与特点

15.1.1 知识产权的概念

知识产权(也称为智慧财产权)是现代社会发展不可缺少的一种法律制度。知识产权保护制度为促进知识的积累与交流,丰富人们的精神生活,提高全民族的科学文化素质,推动经济的发展以及社会进步起到了及其重要的作用。知识产权是指人们基于自己的智力活动创造的成果和经营管理活动中的经验、知识而依法享有的权利。我国《民法通则》规定,知识产权是指民事权利主体(公民、法人)基于创造性的智力成果而享有的权利。

根据有关国际公约规定(世界知识产权组织公约第二条),知识产权的保护对象包括下列各项有关权利:

- 文学、艺术和科学作品;
- 表演艺术家的表演以及唱片和广播节目;
- 人类一切活动领域的发明;
- 科学发现;
- 工业品外观设计;
- 商标、服务标记以及商业名称和标志;
- 制止不正当竞争;
- 在工业、科学、文学艺术领域内由于智力创造活动而产生的一切其他权利。

在世界贸易组织的知识产权协议中,第一部分第一条所规定的知识产权范围还包括“未披露过的信息专有权”,这主要是指工商业经营者所拥有的经营秘密和技术秘密等商业秘密。知识产权保护制度是随着科学技术的进步而不断发展和完善的。随着科学技术的迅速发展,知识产权保护对象的范围不断扩大,不断涌现新型的智力成果,如计算机软件、生物工程技术、遗传基因技术以及植物新品种等,这些都是当今世界各国所公认的知识产权的保护对象。知识产权可分为工业产权和著作权两类。

1. 工业产权

根据保护工业产权巴黎公约第一条的规定,工业产权包括专利、实用新型、工业品外观设计、商标、服务标记、厂商名称、产地标记或原产地名称以及制止不正当竞争等项内

容。此外,商业秘密、微生物技术、遗传基因技术等也属于工业产权保护的客体。近年来,在一些国家可以通过申请专利,对计算机软件进行专利保护。对于工业产权保护的客体,可以分为“创造性成果权利”和“识别性标记权利”。发明、实用新型和工业品外观设计等属于创造性成果权利,它们都表现出比较明显的智力创造性。其中,发明和实用新型是利用自然规律做出的解决特定问题的新的技术方案,工业品外观设计是确定工业品外表的美学创作,完成人需要付出创造性劳动。商标、服务标记、厂商名称、产地标记或原产地名称以及我国反不正当竞争法第五条中规定的知名商品所特有的名称、包装、装潢等为识别性标记权利。

2. 著作权

著作权(也称为版权)是指作者对其创作的作品享有的人身权和财产权。人身权包括发表权、署名权、修改权和保护作品完整权等;财产权包括作品的使用权和获得报酬权,即以复制、表演、播放、展览、发行、摄制电影、电视、录像或者改编、翻译、注释、编辑等方式使用作品的权利,以及许可他人以上述方式使用作品并由此获得报酬的权利。关于著作权保护的客体,按照《保护文学艺术作品伯尔尼公约》第二条规定,包括文学、科学和艺术领域内的一切作品(不论其表现形式或方式如何),诸如书籍、小册子和其他著作,讲课、演讲和其他同类性质作品,戏剧或音乐作品,舞蹈艺术作品和哑剧作品,配乐或未配乐的乐曲,电影作品以及与使用电影摄影艺术类似的方法表现的作品,图画、油画、建筑、雕塑、雕刻和版画,摄影作品以及使用与摄影艺术类似的方法表现的作品,与地理、地形建筑或科学技术有关的示意图、地图、设计图、草图和立体作品等。

有些智力成果可以同时成为这两类知识产权保护的客体。例如,计算机软件和实用艺术品属于著作权保护的同时,权利人还可以通过申请发明专利和外观设计专利,获得专利权,成为工业产权保护的客体。在美国和欧洲的一些国家,如果计算机软件自身包含技术构成,软件又能实现某方面的技术效果,如工业自动化控制等,则不应排除专利保护。按照世界知识产权组织公约,科学发现也被列为知识产权。我国《民法通则》第九十七条规定了科学发现权的法律地位,但很难将其归属于工业产权或著作权。可见新产生的一些知识产权不一定就归为这两个类别。知识产权所保护的客体是依赖人类智力劳动创造的,特别是高科技创新产业迸发出的呈现千姿百态的知识财产,都属于人类智力劳动的成果,法律都赋予它们民事权利。

15.1.2 知识产权的特点

1. 无形性

知识产权是一种无形财产权。知识产权的客体指的是智力创造性成果(也称为知识产品),是一种没有形体的精神财富。它是一种可以脱离其所有者而存在的无形信息,可

以同时为多个主体所使用,在一定条件下不会因多个主体的使用而使该项知识财产自身遭受损耗或者消失。

2. 双重性

某些知识产权具有财产权和人身权双重属性,例如著作权,其财产权属性主要体现在所有人享有的独占权以及许可他人使用而获得报酬的权利,所有人可以通过独自实施获得收益,也可以通过有偿许可他人实施获得收益,还可以像有形财产那样进行买卖或抵押;其人身权属性主要是指署名权等。有的知识产权具有单一的属性,例如,发现权只具有名誉权属性,而没有财产权属性;商业秘密只具有财产权属性,而没有人身权属性;专利权和商标权主要体现为财产权。

3. 确认性

无形的智力创作性成果不像有形财产那样直观可见,因此,智力创作性成果的财产权需要依法审查确认,以得到法律保护。例如,我国的发明人所完成的发明,其实用新型或者外观设计,已经具有价值和使用价值,但是,其完成人并不能自动获得专利权。完成人必须依照专利法的有关规定,向国家专利局提出专利申请;专利局依照法定程序进行审查,对于符合专利法规定条件的,由专利局做出授予专利权的决定,颁发专利证书,只有当专利局发布授权公告后,其完成人才享有该项知识产权。又如,对于商标权的获得,大多数国家(包括中国)都实行注册制。只有向国家商标局提出注册申请,经审查核准注册后,才能获得商标权。文学艺术作品以及计算机软件的著作权虽然是自作品完成后其权利即自动产生,但有些国家也要在实行登记或标注版权标记后才能得到保护。

4. 独占性

由于智力成果具有可以同时被多个主体所使用的特点,因此,法律授予知识产权一种专有权,具有独占性。未经权利人许可,任何单位或个人不得使用,否则就构成侵权,应承担相应的法律责任。法律对各种知识产权都规定了一定的限制,但这些限制不影响其独占性特征。少数知识产权不具有独占性特征,例如技术秘密的所有人不能禁止第三方使用其独立开发完成的或者合法取得的相同技术秘密。可以说,商业秘密不具备完全的财产权属性。

5. 地域性

知识产权具有严格的地域性特点,即各国主管机关依照本国法律授予的知识产权只能在其本国领域内受法律保护。例如,中国专利局授予的专利权或中国商标局核准的商标专用权只能在中国领域内受保护,其他国家则不给予保护。外国人在我国领域外使用中国专利局授权的发明专利,不侵犯我国专利权。所以,我国公民、法人完成的发明创造要想在外国受保护,必须在外国申请专利。著作权虽然自动产生,但它受地域限制。我国法律对外国人的作品并不都给予保护,只保护共同参加国际条约国家的公民的作品。同

样,公约的其他成员国也按照公约规定,对我国公民和法人的作品给予保护。还有的是按照两国的双边协定,相互给予对方国民的作品以保护。

6. 时间性

知识产权具有法定的保护期限,一旦保护期限届满,权利将自行终止,成为社会公众可以自由使用的知识。至于期限的长短,依各国的法律确定。例如,我国发明专利的保护期为 20 年,实用新型专利权和外观设计专利权的期限为 10 年,均自专利申请日起计算。我国公民的作品发表权的保护期为作者终生及其死亡后 50 年。我国商标权的保护期限自核准注册之日起 10 年内有效,但可以根据其所有人的需要无限地续展权利期限。在期限届满前 6 个月内申请续展注册,每次续展注册的有效期为 10 年,续展注册的次数不限。如果商标权人逾期不办理续展注册,其商标权也将终止。商业秘密受法律保护的期限是不确定的,该秘密一旦为公众所知悉,即成为公众可以自由使用的知识。

15.1.3 我国保护知识产权的法规

目前,我国已形成了比较完备的知识产权保护法律体系。保护知识产权的法律主要有:

- 《中华人民共和国著作权法》;
- 《中华人民共和国专利法》;
- 《中华人民共和国继承法》;
- 《中华人民共和国公司法》;
- 《中华人民共和国合同法》;
- 《中华人民共和国商标法》;
- 《中华人民共和国产品质量法》;
- 《中华人民共和国反不正当竞争法》;
- 《中华人民共和国刑法》;
- 《中华人民共和国计算机信息系统安全保护条例》;
- 《中华人民共和国计算机软件保护条例》;
- 《中华人民共和国著作权法实施条例》等。

15.2 计算机软件著作权的主体与客体

15.2.1 计算机软件著作权的主体

计算机软件著作权的主体指享有著作权的人。根据著作权法和《计算机软件保护条

例》的规定,计算机软件著作权的主体包括公民、法人和其他组织。著作权法和《计算机软件保护条例》未规定对主体的行为能力限制,同时对外国人、无国籍人的主体资格,奉行“有条件”的国民待遇原则。

1. 公民

公民(指自然人)通过以下途径取得软件著作权主体资格:

- 公民自行独立开发软件(软件开发者);
- 订立委托合同,委托他人开发软件,并约定软件著作权归自己享有;
- 通过转让途径取得软件著作权财产权主体资格(软件权利的受让者);
- 公民之间或与其他主体之间,对计算机软件进行合作开发而产生的公民群体或者公民与其他主体成为计算机软件作品的著作权人;
- 根据《继承法》的规定通过继承取得软件著作权财产权主体资格。

2. 法人

法人是具有民事权利能力和民事行为能力,依法独立享有民事权利和承担义务的组织。计算机软件的开发往往需要较大投资和较多的人员,法人则具有资金来源丰富和科技人才众多的优势,因而法人是计算机软件著作权的重要主体。法人取得计算机软件著作权主体资格一般通过以下途径:

- 由法人组织并提供创作物质条件所实施的开发,并由法人承担社会责任;
- 通过接受委托、转让等各种有效合同关系而取得著作权主体资格;
- 因计算机软件著作权主体(法人)发生变更而依法成为著作权主体。

3. 其他组织

其他组织是指除去法人以外的能够取得计算机软件著作权的其他民事主体。包括非法人单位、合作伙伴等。

15.2.2 计算机软件著作权的客体

计算机软件著作权的客体是指著作权法保护的计算机软件著作权的范围(受保护的对象)。根据《著作权法》第三条和《计算机软件保护条例》第二条的规定,著作权法保护的计算机软件是指计算机程序及其有关文档。著作权法对计算机软件的保护是指计算机软件的著作权人或者其受让者依法享有著作权的各项权利。

1. 计算机程序

《根据计算机软件保护条例》第三条第一款的规定,计算机程序是指为了得到某种结果而可以由计算机等具有信息处理能力的装置执行的代码化指令序列,或者可被自动转换成代码化指令序列的符号化语句序列。计算机程序包括源程序和目标程序,同一程序的源程序文本和目标程序文本视为同一软件作品。

2. 计算机软件的文档

根据《计算机软件保护条例》第三条第二款的规定,计算机程序的文档是指用自然语言或者形式化语言所编写的文字资料和图表,用来描述程序的内容、组成、设计、功能规格、开发情况、测试结果及使用方法等。文档一般包括程序设计说明书、流程图、用户手册等。

15.3 计算机软件受著作权法保护的条件

《计算机软件保护条例》规定,依法受到保护的计算机软件作品必须符合下列条件。

1. 独立创作

受保护的软件必须由开发者独立开发,任何复制或抄袭他人开发的软件不能获得著作权。当然,软件的独创性不同于专利的创造性。程序的功能设计往往被认为是程序的思想概念,根据著作权法不保护思想概念的原则,任何人可以设计具有类似功能的另一件软件作品。但如果用了他人软件作品的逻辑步骤的组合方式,则对他人软件构成侵权。

2. 可被感知

受著作权法保护的作品应当是作者创作思想在固定载体上的一种实际表达。如果作者的创作思想未表达出来或不可以被感知,就不能得到著作权法的保护。因此,《计算机软件保护条例》规定,受保护的软件必须固定在某种有形物体上,例如固定在存储器、磁盘、磁带等设备上,也可以是其他有形物,如纸张等。

3. 逻辑合理

逻辑判断功能是计算机系统的基本功能。因此,受著作权法保护的计算机软件作品必须具备合理的逻辑思想,并以正确的逻辑步骤表现出来,才能达到软件的设计功能。毫无逻辑性的计算机软件,不能计算出正确结果,也就毫无价值。

根据《计算机软件保护条例》第六条的规定,除计算机软件的程序和文档外,著作权法不保护计算机软件开发所用的思想、概念、发现、原理、算法、处理过程和运算方法。也就是说利用已有的上述内容开发软件,并不构成侵权。因为开发软件时所采用的思想、概念等均属计算机软件基本理论的范围,是设计开发软件不可或缺的理论依据,属于社会公有领域,不能为个人专有。

15.4 计算机软件著作权的权利

15.4.1 计算机软件的著作人身权

《中华人民共和国著作权法》规定,软件作品享有两类权利,一类是软件著作权的人身

权(精神权利);另一类是软件著作权的财产权(经济权利)。《计算机软件保护条例》规定,软件著作权人享有发表权和开发者身份权,这两项权利与软件著作权人的人身权是不可分离的。

1. 发表权

发表权是指决定软件作品是否公之于众的权利,即指软件作品完成后,以复制、展示、发行或者翻译等方式使软件作品在一定数量的不特定人的范围内公开。发表权的具体内容包括软件作品发表的时间、发表的形式以及发表的地点等。

2. 开发者身份权(也称为署名权)

开发者身份权是指作者为表明身份在软件作品中署自己名字的权利。署名可有多种形式,既可以署作者的姓名,也可以署作者的笔名。对一部作品来说,通过署名即可对作者的身份给予确认。我国著作权法规定,如无相反证明,在作品上署名的公民、法人或非法人单位为作者。因此,作品的署名对确认著作权的主体具有重要意义。开发者的身份权,不随软件开发者的消亡而丧失,且无时间限制。

15.4.2 计算机软件的著作财产权

著作权中的财产权是指能够给著作权人带来经济利益的权利。这种经济利益的实现,要依靠著作权人对作品的使用才能获得。财产权通常是指由软件著作权人控制和支配,并能够为权利人带来一定经济效益的权利。《计算机软件保护条例》规定,软件著作权人享有下述软件财产权。

(1) 使用权:即在不损害社会公共利益的前提下,以复制、修改、发行、翻译、注释等方式使用软件的权利。

(2) 复制权:即将软件作品制作一份或多份的权利。复制权就是版权所有人决定实施或不实施上述复制行为,或者禁止他人复制其受保护作品的权利。

(3) 修改权:即对软件进行增补、删节或者改变指令、语句顺序等以提高、完善原软件作品的权利。修改权即指作者享有的修改或者授权他人修改软件作品的权利。

(4) 发行权:发行指为满足公众的合理需求,通过出售、出租等方式向公众提供一定数量的作品复制件的权利。发行权即以出售或赠与方式向公众提供软件的原件或者复制件的权利。

(5) 翻译权:是指以不同于原软件作品的一种程序语言转换该作品原使用的程序语言,而重现软件作品内容的创作权利。简单地说,也就是指将原软件从一种程序语言转换成另一种程序语言的权利。

(6) 注释权:是指对软件作品中的程序语句进行解释,以便更好地理解软件作品的权利。注释权即著作权人对自己的作品享有进行注释的权利。

(7) 信息网络传播权：以有线或者无线信息网络方式向公众提供软件作品，使公众可在其个人选定的时间和地点获得软件作品的权利。

(8) 出租权：即有偿许可他人临时使用计算机软件的复制件的权利，但计算机软件不是出租的主要标的的除外。

(9) 使用许可权和获得报酬权：即许可他人以上述方式使用软件作品的权利（许可他人行使软件著作权中的财产权）和依照约定或者有关法律规定获得报酬的权利。

(10) 转让权：即向他人转让软件的使用权和使用许可权的权利。软件著作权人 can 以全部或者部分转让软件著作权中的财产权。

15.4.3 软件合法持有人的权利

根据《计算机软件保护条例》的规定，软件合法复制品的所有人享有下列权利：

- 根据使用的需要把软件装入计算机等能存储信息的装置内；
- 根据需要进行必要的复制；
- 为了防止复制品损坏而制作备份复制品。这些复制品不得以任何方式提供给他人使用；而且，在所有人丧失该合法复制品所有权时，要负责将备份复制品销毁；
- 为了把该软件用于实际的计算机应用环境或者改进其功能性能而进行必要的修改。但是，除合同约定外，未经该软件著作权人许可，不得向任何第三方提供修改后的软件。

15.4.4 计算机软件著作权的行使

1. 软件经济权利的许可使用

软件经济权利的许可使用是指软件著作权人或权利合法受让者，通过合同方式许可他人使用其软件，并获得报酬的一种软件贸易形式。许可使用的方式可分为以下几种：

(1) 独占许可使用：权利人通过书面合同授权，被授权方可以根据合同规定的方式、条件和时间确定独占性，权利人不得将软件使用权授予第三方，权利人自己不能使用该软件。

(2) 独家许可使用：权利人通过书面合同授权，被授权方可以根据合同规定的方式、条件和时间确定独占性，权利人不得将软件使用权授予第三方，权利人自己可以使用该软件。

(3) 普通许可使用：权利人通过书面合同授权，被授权方可以根据合同规定的方式、条件和时间确定独占性，权利人可以将软件使用权授予第三方，权利人自己可以使用该软件。

(4) 法定许可使用和强制许可使用：即在法律特定的条款下，不经软件著作权人许

可使用其软件。

2. 软件经济权利的转让使用

软件经济权利的转让使用是指软件著作权人将其享有的软件著作权中的经济权利全部转移给他人。软件经济权利的转让将改变软件权利的归属,原始著作权人的主体地位随着转让活动的发生而丧失,软件著作权受让者成为新的著作权主体。《计算机软件保护条例》规定,软件著作权转让必须签订书面合同。同时,软件转让活动不能改变软件的保护期。转让方式包括卖出、赠与、抵押、赔偿等,可以定期转让或者永久转让。

15.4.5 计算机软件著作权的保护期

根据《著作权法》和《计算机软件保护条例》的规定,计算机软件著作权的权利自软件开发完成之日起产生,保护期为 50 年。保护期满,除开发者身份权以外,其他权利终止。一旦计算机软件著作权超出保护期,软件就进入公有领域。计算机软件著作权人的单位终止和计算机软件著作权人的公民死亡均无合法继承人时,除开发者身份权以外,该软件的其他权利进入公有领域。软件进入公有领域后成为社会公共财富,公众可无偿使用。

15.5 计算机软件著作权的归属

15.5.1 软件著作权归属的基本原则

我国著作权法对著作权的归属采取了“创作主义”原则,明确规定著作权属于作者,除非另有规定。《计算机软件保护条例》第九条规定“软件著作权属于软件开发者,本条例另有规定的情况除外。”这是我国计算机软件著作权归属的基本原则。

计算机软件开发者是计算机软件著作权的原始主体,也是享有权利最完整的主体。软件作品是开发者从事智力创作活动所取得的智力成果,是脑力劳动的结晶。其开发创作行为使开发者直接取得该计算机软件的著作权。因此,《计算机软件保护条例》第九条明确规定“软件著作权属于软件开发者”,即以软件开发的事实来确定著作权的归属,谁完成了计算机软件的开发工作,该软件的著作权就归谁享有,《计算机软件保护条例》另有规定的除外。

15.5.2 职务开发软件著作权的归属

职务软件作品是指公民在单位任职期间为执行本单位工作任务所开发的计算机软件作品。《计算机软件保护条例》第十三条做出了明确的规定,公民在单位任职期间所开发的软件,如果是执行本职工作的结果,即针对本职工作中明确指定的开发目标所开发的,

或者是从事本职工作活动所预见的结果或自然的结果,则该软件的著作权属于该单位;如果主要使用了单位的资金、专用设备、未公开的专门信息等物资技术条件所开发并由法人或者其他组织承担责任的软件,该软件的著作权也归单位所有。

对于公民在非职务期间创作的计算机程序,其著作权属于某项软件作品的开发单位,还是直接开发软件作品的个人,可按照《计算机软件保护条例》第十三条规定的 3 条标准确定。

1. 所开发的软件作品不是执行其本职工作的结果

任何受雇于一个单位的人员,都会被安排在一定的工作岗位和分派相应的工作任务。完成分派的工作任务就是他的本职工作。本职工作的直接成果也就是其完成工作任务的结果。当然具体工作成果又会产生许多效益、产生范围更广的结果。但该条标准指的是雇员本职工作最直接的成果。若雇员开发的软件不是执行本职工作的结果,则构成非职务计算机软件著作权的条件之一。

2. 开发的软件作品与开发者在单位中从事的工作内容无直接联系

如果雇员在单位担任软件开发工作,引起争议的软件作品与其本职工作中明确指定的开发目标无关,软件作品的内容也与其本职工作所开发的软件的功能、逻辑思维和重要数据无关,即该雇员所开发的软件作品与其本职工作没有直接的关系,则构成非职务计算机软件著作权的第二个条件。

3. 开发的软件作品未利用单位的物质技术条件

开发创作软件作品所使用的物质技术条件,即开发软件作品所必须的设备、数据、资金和其他软件开发环境不属于雇员所在的单位所有。没有使用受雇单位的任何物质技术条件构成非职务软件著作权的第三个条件。

雇员进行本职工作以外的软件开发创作,必须同时符合上述 3 个条件,才能算是非职务软件作品,雇员个人才享有软件著作权。常有软件开发符合前两个条件,但利用了单位的技术情报资料以及计算机设备等物质技术条件的情况。此时较好的解决方法是由单位和雇员双方协商确定。如对于公民在非职务期间利用单位物质条件开发的与单位业务范围无关的计算机程序,其著作权可以属于创作程序的作者,但作者许可第三方使用软件时,应当支付单位合理的物质条件使用费,如计算机机时费等。若通过协商不能解决,应由有关部门按上述 3 条标准做出裁定。

15.5.3 合作开发软件著作权的归属

合作开发软件是指两个或两个以上公民、法人或其他组织订立协议,共同参加某项计算机软件的开发并分享软件著作权的形式。《计算机软件保护条例》第十条规定:“由两个以上的自然人、法人或者其他组织合作开发的软件,其著作权的归属由合作开发者签订

书面合同约定。无书面合同或者合同未作明确约定,合作开发的软件可以分割使用的,开发者对各自开发的部分可以单独享有著作权;但是,行使著作权时,不得扩展到合作开发的软件整体的著作权。合作开发的软件不能分割使用的,其著作权由合作开发者共同享有,通过协商一致行使;如不能协商一致,又无正当理由,任何一方不得阻止他方行使除转让权以外的其他权利,但是所得收益应合理分配给所有合作开发者。”根据此规定,对合作开发软件著作权的归属应掌握以下4点。

(1) 由两个以上的单位或公民共同开发完成的软件属于合作开发的软件。对于合作开发的软件,其著作权的归属一般是由各合作开发者共同享有;但如果有关软件著作权的协议,则按照协议确定软件著作权的归属。

(2) 鉴于合作开发软件的著作权由两个以上单位或者个人共同享有,因而为了避免在软件著作权的行使中产生纠纷,规定“合作开发的软件,其著作权的归属由合作开发者签订书面合同约定”。

(3) 对于合作开发的软件著作权按以下规定执行:“无书面合同或者合同未作明确约定,合作开发的软件可以分割使用的,开发者对各自开发的部分可以单独享有著作权;但是,行使著作权时,不得扩展到合作开发的软件整体的著作权。合作开发的软件不能分割使用的,其著作权由合作开发者共同享有,通过协商一致行使;如不能协商一致,又无正当理由,任何一方不得阻止他方行使除转让权以外的其他权利,但是所得收益应合理分配给所有合作开发者。”

(4) 合作开发者对于软件著作权中的转让权不得单独行使。因为转让权的行使将涉及软件著作权权利主体的改变,所以软件的合作开发者在行使转让权时,必须与各合作开发者协商,在征得同意的情况下方能行使该项专有权利。

15.5.4 委托开发的软件著作权归属

委托开发的软件作品属于著作权法规定的委托软件作品。委托开发软件作品著作权关系的建立,一般由委托方与受委托方订立合同而成立。委托开发软件作品中,委托方的责任主要是提供资金、设备等物质条件,并不直接参与开发软件作品的活动。受托方的主要责任是根据委托合同规定的目标开发出符合条件的软件。关于委托开发软件著作权的归属,《计算机软件保护条例》第十一条规定:“接受他人委托开发的软件,其著作权的归属由委托者与受委托者签订书面合同约定;无书面合同或者合同未作明确约定的,其著作权由受托人享有。”根据该规定,委托开发的软件著作权的归属按以下标准确定。

(1) 委托开发软件作品系根据委托方的要求,由委托方与受托方以合同确定的权利和义务的关系而进行开发的软件。因此,软件作品著作权归属应当作为合同的重要条款予以明确约定。对于当事人已经在合同中约定软件著作权归属关系的,如事后发生纠纷,

软件著作权的归属仍应当根据委托开发软件的合同来确定。

(2) 若在委托开发软件活动中,委托者与受委托者没有签订书面协议,或者在协议中未对软件著作权归属做出明确的约定,则软件著作权属于受委托者,即属于实际完成软件的开发者。

15.5.5 接受任务开发的软件著作权归属

根据社会经济发展的需要,对于一些涉及国家基础项目或者重点设施的计算机软件,往往采取由政府有关部门或上级单位下达任务的方式,完成软件的开发工作。对于下达任务开发的软件,其著作权的归属关系,《计算机软件保护条例》第十二条做出了明确的规定:“由国家机关下达任务开发的软件,著作权的归属与行使由项目任务书或者合同规定;项目任务书或者合同中未作明确规定,软件著作权由接受任务的法人或者其他组织享有。”根据该规定,国家或上级下达任务开发的软件著作权归属应按以下两条标准确定。

(1) 下达任务开发的软件著作权的归属关系,首先应以项目任务书的规定或者双方的合同约定为准。

(2) 下达任务的项目任务书或者双方订立的合同中未对软件著作权归属做出明确的规定或者约定的,其软件著作权属于接受并实际完成开发软件任务的单位。

15.5.6 计算机软件著作权主体变更后软件著作权的归属

计算机软件著作权的主体,因一定的法律事实而发生变更。如作为软件著作权人的公民的死亡,单位的变更,软件著作权的转让以及人民法院对软件著作权的归属做出裁判等。软件著作权主体的变更必然引起软件著作权归属的变化。对此,《计算机软件保护条例》也作了一些规定。因计算机软件主体变更引起的权属变化有以下几种:

1. 公民继承的软件权利归属

《计算机软件保护条例》第十五条规定:“在软件著作权的保护期内,软件著作权的继承者可根据《中华人民共和国继承法》的有关规定,继承本条例第八条规定的除署名权以外的其他权利。”按照该规定,软件著作权的合法继承人依法享有继承被继承人享有的软件著作权的使用权、使用许可权和获得报酬权等权利。继承权的取得和继承顺序的排定等均按照继承法的规定进行。

2. 单位变更后软件权利归属

《计算机软件保护条例》第十五条规定:“软件著作权属于法人或其他组织的,法人或其他组织变更、终止后,其著作权在本条例规定的保护期内由承受其权利义务的法人或其他组织享有;”按照该规定,作为软件著作权人的单位发生变更(如单位的合并、破产等),而其享有的软件著作权仍处在法定的保护期限内,可以由合法的权利承受单位享有原始

著作权人所享有的各项权利。依法承受软件著作权的单位,成为该软件的后续著作权人,可在法定的条件下行使所承受的各项专有权利。一般认为,“各项权利”包括署名权等著作人身权在内的全部权利。

3. 权利转让后软件著作权归属

《计算机软件保护条例》第二十条规定:“转让软件著作权的,当事人应当订立书面合同。”计算机软件作品财产权按照该规定发生转让后,必然引起著作权主体的变化,产生新的软件著作权归属关系。软件权利的转让应当根据我国有关法规以签订、执行书面合同的方式进行。软件权利的受让者可依法行使其享有的权利。

4. 司法判决、裁定引起的软件著作权归属问题

计算机软件著作权是公民、法人和其他组织享有的一项重要的民事权利。因而在民事权利行使和流转的过程中,难免会发生涉及计算机软件著作权作为标的物的民事及经济关系,也难免发生争议和纠纷。争议和纠纷发生后由人民法院的民事判决、裁定而产生软件著作权主体的变更,也会产生软件著作权归属问题。因司法裁判引起软件著作权的归属问题主要有4类:第一类是由人民法院对著作权归属纠纷中权利的最终归属做出司法裁判,从而变更了计算机软件著作权原有归属;第二类是计算机软件的著作权人为民事法律关系中的债务人(债务形成的原因可能多种多样,如合同关系或者损害赔偿关系等),人民法院将其软件著作财产权判归债权人享有;第三类是人民法院做出民事判决判令软件著作权人履行民事给付义务,在判决生效后的执行程序中,因无其他财产可供执行,将软件著作财产权执行给对方折抵债务;第四类是根据破产法的规定,软件著作权人破产还债,软件著作财产权作为法律规定的破产财产构成的“其他财产权利”,作为破产财产由人民法院判决分配。

5. 保护期限届满权利丧失

软件著作权的法定保护期限可以确定计算机软件的主体能否依法变更。如果软件著作权已过保护期,该软件进入公有领域,便丧失了专有权,也就没有必要改变权利主体了。根据软件保护条例的规定,计算机软件著作权主体变更必须在该软件著作权的保护期限内进行,转让活动的发生不改变该软件著作权的保护期。这也就是说,转让活动也不能延长该软件著作权的保护期限。

15.6 计算机软件著作权侵权的鉴别

侵犯计算机软件著作权的违法行为的鉴别,主要依据保护知识产权的相关法律来判断。违反著作权法和《计算机软件保护条例》等法律禁止的行为,便是侵犯计算机著作权的违法行为,这是鉴别违法行为的根本原则。对于法律规定不禁止,也不违反相关法律基

本原则的行为,不认为是违法行为。在法律无明文规定的情况下,违背著作权法和《计算机软件保护条例》等法律的基本原则,以及社会主义公共生活准则和社会道德规范的行为,也应该视为违法行为。在一般情况下,损害他人著作财产权或人身权的行为,总是违法行为。

15.6.1 计算机软件著作权侵权行为

根据《计算机软件保护条例》第二十三条的规定,凡是行为人主观上具有故意或者过失对著作权法和《计算机软件保护条例》保护的计算机软件人身权和财产权实施侵害行为的,都构成计算机软件的侵权行为。该规定定性的侵犯计算机软件著作权的情况,是认定软件著作权侵权行为的法律依据。计算机软件著作权侵权行为主要有以下几种。

1. 未经软件著作权人的同意而发表或者登记其软件作品

软件著作权人享有软件作品的公开发表权,未经允许著作权人以外的其他任何人都无权擅自发表特定的软件作品。如果实施这种行为,就构成侵犯著作权人的发表权。

2. 将他人开发的软件当作自己的作品发表或者登记

此种行为主要侵犯了软件著作权的开发者身份权和署名权。侵权行为人欺世盗名,剽窃软件开发者的劳动成果,将他人开发的软件作品假冒为自己的作品而署名发表。只要行为人实施了这种行为,不管其发表该作品是否经过软件著作权人的同意,都构成侵权。

3. 未经合作者的同意将与他人合作开发的软件当作自己独立完成的作品发表或者登记

此种侵权行为发生在软件作品的合作开发者之间。作为合作开发的软件,软件作品的开发者身份为全体开发者,软件作品的发表权也应由全体开发者共同行使。如果未经其他开发者同意,将合作开发的软件当作自己的独创作品发表,即构成本条规定的侵权行为。

4. 在他人开发的软件上署名或者更改他人开发的软件上的署名

这种行为是指在他人开发的软件作品上添加自己的署名,或者替代软件开发者署名以及或者将软件作品上开发者的署名进行更改的行为。这种行为侵犯了软件著作权人的开发者身份权及署名权。此种行为与第2条规定行为的区别主要是对已发表的软件作品实施的行为。

5. 未经软件著作权人或者其合法受让者的许可,修改或翻译其软件作品

此种行为是侵犯了著作权人或其合法受让者的使用权中的修改权、翻译权。对不同版本的计算机软件,新版本往往是旧版本的提高和改善。这种提高和改善实质上是对原软件作品的修改、演绎。此种行为应征得软件作品原版本著作权人的同意,否则构成侵权。如果征得软件作品著作权人的同意,因修改和改善而新增加的部分,创作者应享有著

作权。

6. 未经软件著作权人或其合法受让者的许可,复制或部分复制其软件作品

此种行为侵犯了著作权人或其合法受让者的使用权中的复制权。计算机软件的复制权是计算机软件最重要的著作财产权,也是通常计算机软件侵权行为的对象。这是由于软件载体价格相对低廉,复制软件简单易行效率极高,而销售非法复制的软件即可获得高额利润。因此,复制是常见的侵权行为,是防止和打击的主要对象。当软件著作权经当事人的约定合法转让给受让者以后,软件开发未经允许不得复制该软件,否则也构成本条规定的侵权行为。

7. 未经软件著作权人及其合法受让者同意,向公众发行或出租其软件的复制品

此种行为侵犯了著作权人或其合法受让者的发行权与出租权。

8. 未经软件著作权人或其合法受让者同意,向任何第三方办理软件权利

许可或转让事宜

这种行为侵犯了软件著作权人或其合法受让者的使用许可权和转让权。

9. 未经软件著作权人及其合法受让者同意,通过信息网络传播著作权人的软件

这种行为侵犯了软件著作权人或其合法受让者的信息网络传播权。

10. 共同侵权行为

二人以上共同实施《计算机软件保护条例》第二十三条和第二十四条规定的侵权行为,构成共同侵权行为。对行为人并没有实施《计算机软件保护条例》第二十三条和第二十四条规定的行为,但实施了给侵权行为人进行侵权活动提供设备、场所或解密软件,或者为侵权复制品提供仓储、运输条件等行为,也构成共同侵权行为。共同侵权行为指在行为人之间具有共同的故意或过失侵权行为,其构成的条件有两个,一是行为人的过错是共同的,而不论行为人的行为在整个侵权过程中所起的作用如何;二是行为人主观上要有故意或过失的过错。如果具备这两个条件,各个行为人实施的侵权行为虽然各不相同,也同样构成共同侵权。两个条件如果缺乏一个,则不构成共同侵权,或者不构成任何侵权。

15.6.2 不构成计算机软件侵权的合理使用行为

我国《计算机软件保护条例》第八条第四项和第十六条规定,获得使用权或使用许可权(视合同条款)后,可以对软件进行复制而无须通知著作权人,亦不构成侵权。合法持有软件复制品的单位、公民在未经著作权人同意的情况下,亦享有复制与修改权。合法持有软件复制品的单位、公民,在未经软件著作权人同意的情况下,可以根据自己使用的需要将软件装入计算机,为了存档也可以制作备份复制品,为了把软件用于实际的计算机环境或者改进其功能时也可以进行必要的修改,但是备份制品和修改后的文本不能以任何方式提供给他人,超过以上权利,即视为侵权行为。区分合理使用与非合理使用的判别标准

一般有：

- 软件作品是否合法取得。这是合理使用的基础；
- 使用目的是否具有商业营业性，如果使用的目的是为商业性营利，就不属合理使用的范围；
- 合理使用一般为少量的使用，所谓少量的界限应根据其使用的目的，以行业惯例和人们的一般常识综合确定。超过通常认为的少量界限，即可认为不属于合理使用。

我国《计算机软件保护条例》第十七规定：“为了学习和研究软件内含的设计思想和原理，通过安装、显示、传输或者存储软件的方式使用软件的，可以不经软件著作权人许可，不向其支付报酬。”

15.6.3 计算机著作权软件侵权的识别

计算机软件明显区别于其他著作权法保护的客体，它具有以下特点。

1. 技术性

计算机软件的技术性是指其创作和开发的高技术性。具有一定规模的软件的创作和开发，一般难度大、周期长、投资高，需要良好组织、严密管理且各方面人员配合协作，并借助现代化技术和高科技工具。

2. 依赖性

计算机程序的依赖性是指人们对其的感知依赖于计算机的特性。著作权保护的其他作品一般都可以依赖人的感觉器官直接感知。但计算机程序则不能被人们直接感知，它的内容只能依赖计算机等专用设备才能充分表现出来，从而被人们感知。

3. 多样性

计算机程序的多样性是指计算机程序表达的多样性。计算机程序的表达较著作权法保护的其他对象特殊，其既能以源代码表达，还可以以目标代码和微码等表达，表达形式多样。计算机程序的存储媒体也多种多样，同一种程序可以存储在纸张、磁盘、磁带、光盘和集成电路上。计算机程序的载体大多数精巧灵便。此外，计算机程序的内容与表达难以严格区别界定。

4. 运行性

计算机程序的运行性是指计算机程序功能的运行性。计算机程序不同于一般的文字作品，它主要的功能在于使用。也就是说计算机程序的功能只能通过对程序的使用、运行才能充分体现出来。计算机程序采用数字化形式存储、转换，复制品与原作品一般无明显区别。

根据计算机软件的特点，对计算机软件侵权行为的识别可以将发生争议的某一计算

机程序与比照物(权利明确的正版计算机程序)进行对比和鉴别,从两个软件的相似性或完全相同来判断,做出侵权认定。软件作品常常表现为计算机程序的不惟一性,两个运行结果相同的计算机程序,或者两个计算机软件的源代码程序不相似或不完全相似,前者不一定构成侵权,而后者不一定不构成侵权。识别侵权(盗版)软件可以采取下列简单方法和步骤:

① 对被识别的软件与正版软件直接进行目录、文件名对比以及部分内容对比。如果这两者完全一致,就可以初步认定没有手续而拥有该软件的使用者或销售者为软件侵权者;如果并非完全一致,而是大部分一致,就需要继续进行识别。

② 对两套软件同时或先后进行安装,观察其安装过程中的屏幕显示,包括软件信息以及使用功能键后的屏幕显示等是否相同。如果相同,则可认定这两套软件的安装手段一致。

③ 对其安装成功后的目录,以及各文件信息进行对比。观察屏幕显示,通过表观现象,包括文件名、文件长度、文件建立(或修改)时间及文件属性 4 个部分进行识别。一般情况下,侵权销售者经过修改的盗版软件与正版软件可能不完全一致,只是修改少部分,而绝大部分文件的表观现象都是一致的。

④ 对盗版软件与正版软件的使用过程进行对比,即对使用过程中的屏幕显示、功能、功能键及使用方法等进行对比。

⑤ 对盗版软件与正版软件源程序进行对比。软件开发者一般不向外公布其源程序,计算机软件以源程序方式向外传播的情况较少,大多是以目标程序的形式向外传播。

15.7 软件著作权侵权的法律责任

当侵权人侵害他人的著作权、财产权或著作人身权,造成权利人财产或非财产的损失,侵权人不履行赔偿义务,法律将强制侵权人承担赔偿责任的民事责任。

1. 民事责任

侵犯计算机著作权以及有关权益的民事责任是指公民、法人或其他组织因侵犯著作权发生的后果依法应承担的法律责任。我国《计算机软件保护条例》第二十三条规定了侵犯计算机著作权的民事责任,即侵犯著作权或者与著作权有关权利的侵权人应当按照权利人的实际损失给予赔偿;实际损失难以计算的,可以按照侵权人的违法所得给予赔偿。赔偿数额还应当包括权利人为制止侵权行为所支付的合理开支。权利人的实际损失或者侵权人的违法所得不能确定的,由人民法院根据侵权行为的情节,判决给予 50 万元以下的赔偿。有下列侵权行为的,应当根据情况,承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等民事责任:

- 未经软件著作权人许可发表或者登记其软件的；
- 将他人软件当作自己的软件发表或者登记的；
- 未经合作者许可，将与他人合作开发的软件当作自己单独完成的作品发表或者登记的；
- 在他人软件上署名或者涂改他人软件上的署名的；
- 未经软件著作权人许可，修改、翻译其软件的；
- 其他侵犯软件著作权的行为。

2. 行政责任

我国《计算机软件保护条例》第二十四条规定了相应的行政责任，即对侵犯软件著作权的行为，著作权行政管理部门应当责令侵权者停止违法行为，没收非法所得，没收、销毁侵权复制品，并可处以每件 100 元或者货值金额 2~5 倍的罚款。有下列侵权行为的，应当根据情况，承担停止侵害、消除影响、公开赔礼道歉、赔偿损失等行政责任：

- 复制或者部分复制著作权人软件的；
- 向公众发行、出租、通过信息网络传播著作权人的软件的；
- 故意避开或者破坏著作权人为保护其软件而采取的技术措施的；
- 故意删除或者改变软件权利管理电子信息的；
- 许可他人行使或者转让著作权人的软件著作权的。

3. 刑事责任

侵权行为触犯刑律的，侵权者应当承担刑事责任。我国《刑法》第二百一十七条、二百一十八条和二百二十条规定，构成侵犯著作权罪及销售侵权复制品罪的，由司法机关追究刑事责任。

15.8 计算机软件的商业秘密权

15.8.1 商业秘密的概念

关于商业秘密的法律保护，各国采取不同的立法。有的制定单行法，有的在反不正当竞争法中规定，有的适用一般侵权行为法。我国的《反不正当竞争法》涉及了商业秘密的保护问题。

1. 商业秘密的定义

在《反不正当竞争法》中商业秘密被定义为“不为公众所知悉的、能为权利人带来经济利益、具有实用性并经权利人采取保密措施的技术信息和经营信息”，经营秘密和技术秘密是商业秘密的基本内容。经营秘密即未公开的经营信息，是指与生产经营销售活动有

关的经营方法、管理方法、产销策略、货源情报、客户名单、标底和标书内容等专有知识；技术秘密即未公开的技术信息，是指与产品生产和制造有关的技术诀窍、生产方案、工艺流程、设计图纸、化学配方、技术情报等专有知识。

2. 商业秘密的构成条件

商业秘密的构成条件是：商业秘密必须具有未公开性，即不为公众所知悉；商业秘密必须具有实用性，即能为权利人带来经济效益；商业秘密必须具有保密性，即采取了保密措施。

3. 商业秘密权

商业秘密是一种无形的信息财产。与有形财产相区别，商业秘密不占据空间，不易为权利人所控制，不发生有形损耗，其权利是一种无形财产权。商业秘密的权利人与有形财产所有权人一样，依法享有占有、使用和收益的权利，即有权对商业秘密进行控制与管理，防止他人采取不正当手段获取与使用；有权依法使用自己的商业秘密，而不受他人干涉；有权通过自己使用或者许可他人使用以至转让所有权，从而取得相应的经济利益；有权处置自己的商业秘密，包括放弃占有、无偿公开、赠与或转让等。

4. 商业秘密的丧失

一项商业秘密受到法律保护的依据，是必须具备上述构成商业秘密的 3 个条件，缺少上述 3 个条件中任何一个都会造成商业秘密丧失保护。

5. 计算机软件与商业秘密

《反不正当竞争法》保护计算机软件，是以计算机软件中是否包含“商业秘密”为必要条件的。而计算机软件是人类知识、智慧、经验和创造性劳动的成果，本身就具有商业秘密的特征，即包含着技术秘密和经营秘密。即使软件尚未开发完成，在软件开发中所形成的知识内容也可构成商业秘密。

15.8.2 计算机软件商业秘密的侵权

侵犯商业秘密，是指行为人（负有约定的保密义务的合同当事人，实施侵权行为的第三方，侵犯本单位商业秘密的行为人）未经权利人（商业秘密的合法控制人）的许可，以非法手段（包括直接从权利人那里窃取商业秘密并加以公开或使用，通过第三方窃取权利人的商业秘密并加以公开或使用）获取计算机软件商业秘密并加以公开或使用的行为。根据我国《反不正当竞争法》第十条的规定，侵犯计算机软件商业秘密的具体表现形式主要有：

（1）以盗窃、利诱、胁迫或其他不正当手段获取权利人的计算机软件商业秘密。盗窃商业秘密，包括单位内部人员盗窃、外部人员盗窃、内外勾结盗窃等手段；以利诱手段获取商业秘密，通常指行为人向掌握商业秘密的人员提供财物或其他优惠条件，诱使其向行为

人提供商业秘密;以胁迫手段获取商业秘密,是指行为人采取威胁、强迫的手段,使他人受强制的情况下提供商业秘密;以其他不正当手段获取商业秘密。

(2) 披露、使用或允许他人使用以不正当手段获取的计算机软件商业秘密。披露是指将权利人的商业秘密向第三方透露或向不特定的其他人公开,使其失去秘密价值;使用或允许他人使用是指非法使用他人商业秘密的具体情形。以非法手段获取商业秘密的行为人,如果将该秘密再行披露或使用,即构成双重的侵权;倘若第三方从侵权人那里获悉了商业秘密而将秘密披露或使用,同样构成侵权。

(3) 违反约定或违反权利人有关保守商业秘密的要求,披露、使用或允许他人使用其所掌握的计算机软件商业秘密。合法掌握计算机软件商业秘密的人,可能是与权利人有合同关系的对方当事人,也可能是权利人单位的工作人员或其他知情人,他们违反合同约定或单位规定的保密义务,将其所掌握的商业秘密擅自公开,或自己使用,或许可他人使用,即构成侵犯商业秘密。

(4) 第三方在明知或应知前述违法行为的情况下,仍然从侵权人那里获取、使用或披露他人的计算机软件商业秘密。这是一种间接的侵权行为。

15.8.3 计算机软件商业秘密侵权的法律责任

根据我国《反不正当竞争法》和《刑法》的规定,计算机软件商业秘密的侵权者将承担行政责任、民事责任以及刑事责任。

1. 侵权者的行政责任

我国《反不正当竞争法》第二十五条规定了相应的行政责任,即对侵犯商业秘密的行为,监督检查部门应当责令侵权者停止违法行为,而后可以根据侵权的情节依法处以 1 万元以上 20 万元以下的罚款。

2. 侵权者的民事责任

计算机软件商业秘密侵权者的侵权行为对权利人的经营造成经济上的损失时,侵权者应当承担经济损害赔偿的民事责任。我国《反不正当竞争法》第二十条规定了侵犯商业秘密的民事责任,即经营者违反该规定,给被侵害的经营者造成损害的,应当承担损害赔偿责任。被侵害的经营者合法权益受到损害的,可以向人民法院提起诉讼。

3. 侵权者的刑事责任

侵权者以盗窃、利诱、胁迫或其他不正当手段获取权利人的计算机软件商业秘密;披露、使用或允许他人使用以不正当手段获取的计算机软件商业秘密;违反约定或违反权利人有关保守商业秘密的要求,披露、使用或允许他人使用其掌握的计算机软件商业秘密,其侵权行为对权利人造成重大损害的,侵权者应当承担刑事责任。我国《刑法》第二百一十九条规定了侵犯商业秘密罪,即实施侵犯商业秘密行为,给商业秘密的权利人造

成重大损失的,处3年以下有期徒刑或者拘役,并处或者单处罚金;造成特别严重后果的,处3年以上7年以下有期徒刑,并处罚金。

15.9 专利权概述

15.9.1 专利权的保护对象与特征

发明创造是产生专利权的基础。发明创造是指发明、实用新型和外观设计,是我国专利法主要保护的客体。我国《专利法实施细则》第2条第1款规定:“专利法所称的发明,是指对产品、方法或者其改进所提出的技术方案”。实用新型(也称小发明)则因国而异,我国《专利法实施细则》第2条第2款规定:“实用新型是指对产品的形状、构造或者其组合所提出的新的技术方案”。外观设计是指对产品的形状、图案、色彩或者它们的结合所做出的富有美感的并适于工业应用的新设计。

专利的发明创造是无形的智力创造性成果,不像有形财产那样直观可见,必须经专利主管机关依照法定程序审查确定。在未经审批以前,任何一项发明创造都不得成为专利。

下列各项属于专利法不适用的对象,因而不授予专利权。

(1) 对违反国家法律、社会公德或者妨害公共利益的发明创造。

(2) 科学发现,即人们通过自己的智力劳动对客观世界已经存在的但未揭示出来的规律、性质和现象等的认识。

(3) 智力活动的规则和方法,即人们进行推理、分析、判断、运算、处理、记忆等思维活动的规则和方法。

(4) 疾病的诊断和治疗方法,即以活的人或者动物为实施对象,并以防病治病为目的,是医护人员的经验体现,而且因被诊断和治疗的对象不同而有区别,不能在工业上应用,不具有实用性。

(5) 动物、植物品种。但动物、植物品种的生产方法,可以依照专利法规定授予专利权。

(6) 用原子核变换方法获得的物质,即用核裂变或核聚变方法获得的单质或化合物。

15.9.2 授予专利权的条件

授予专利权的条件是指一项发明创造获得专利权应当具备的实质性条件。一项发明或者实用新型获得专利权的实质条件为新颖性、创造性和实用性。

1. 新颖性

新颖性是指在申请日以前没有同样的发明或实用新型在国内外出版物公开发表过,

没有在国内公开使用过或以其他方式为公众所知,也没有同样的发明或实用新型由他人向专利局提出过申请并且记载在申请日以后公布的专利申请文件中。在某些特殊情况下,尽管申请专利的发明或者实用新型在申请日或者优先权日之前公开,但在一定的期限内提出专利申请的,仍然具有新颖性。我国专利法规定申请专利的发明创造在申请日以前6个月内,有下列情况之一的,不丧失新颖性:

- 在中国政府主办或者承认的国际展览会上首次展出的;
- 在规定的学术会议或者技术会议上首次发表的;
- 他人未经申请人同意而泄露其内容的。

2. 创造性

创造性是指同申请日以前已有的技术相比,该发明有突出的实质性特点和显著的进步,该实用新型有实质性特点和进步。例如,申请专利的发明解决了人们渴望解决但一直没有解决的技术难题;申请专利的发明克服了技术偏见;申请专利的发明取得了意想不到的技术效果;申请专利的发明在商业上获得成功。一项发明专利是否具有创造性,前提是该项发明是否具备新颖性。

3. 实用性

实用性是指该发明或者实用新型能够制造或者使用,并且能够产生积极的效果,即不会造成环境污染以及能源或者资源的严重浪费,不会损害人体健康。如果申请专利的发明或者实用新型缺乏技术手段,申请专利的技术方案违背自然规律,或利用独一无二的自然条件所完成的技术方案,则不具有实用性。

我国专利法规定,外观设计获得专利权的实质条件为新颖性和美观性。新颖性是指申请专利的外观设计与申请日以前已经在国内外出版物上公开发表的外观设计不相同或者不近似,与申请日前已在国内公开使用过的外观设计不相同或者不近似。美观性是指外观设计用在产品上时能使人产生一种美感,增加产品对消费者的吸引力。

15.9.3 专利的申请

1. 专利申请权

专利申请权是公民、法人或者其他组织依据法律规定或者合同约定享有的就发明创造向专利局提出专利申请的权利。一项发明创造产生的专利申请权归谁所有,主要分为由法律直接规定的情况和依合同约定的情况。专利申请权可以转让,不论专利申请权在哪一个时间段转让,原专利申请人便因此丧失专利申请权,由受让人获得相应的专利申请权。专利申请权可以被继承或赠与。如专利申请人死亡后,其依法享有的专利申请权可以作为遗产,由其合法继承人继承。

2. 专利申请人

专利申请人是指对某项发明创造依法律规定或者合同约定享有专利申请权的公民、法人或者其他组织。专利申请人包括：职务发明创造的单位；非职务发明创造的发明人或者设计人；共同发明创造的发明人或设计人，或者其所属单位；委托发明创造的专利申请人；合同约定的人；受让人。

3. 专利申请的原则

专利申请人及其代理人在办理各种手续时都应当采用书面形式。一份专利申请文件只能就一项发明创造提出专利申请，即“一份申请一项发明”原则。两个或者两个以上的人分别就同样的发明创造申请专利的，专利权授给最先申请人。

4. 专利申请文件

发明或者实用新型申请文件包括请求书、说明书、说明书摘要及权利要求书。外观设计专利申请文件包括请求书、图片或照片。

5. 专利申请日

专利申请日（也称关键日）是专利局或者专利局指定的专利申请受理代办处收到完整专利申请文件的日期。如果申请文件是邮寄的，以寄出的邮戳日为申请日。

6. 专利申请的审批

专利局收到发明专利申请后，一个必要的程序是初步审查。经初步审查认为符合本法要求的，自申请日起满 18 个月，即行公布（公布申请）。专利局可根据申请人的请求，早日公布其申请。自申请日起 3 年内，专利局可以根据申请人随时提出的请求，对其申请进行实质审查。实质审查是专利局对申请专利的发明的新颖性、创造性和实用性等依法进行审查的法定程序。

我国专利法规定：“实用新型和外观设计专利申请经初步审查没有发现驳回理由的，专利局应当做出授予实用新型专利权或者外观设计专利权的决定，发给相应的专利证书，并予以登记和公布”。由此规定可知，对实用新型和外观设计专利申请只进行初步审查，不进行实质审查。

7. 申请权的丧失与恢复

专利法及其实施细则有许多条款规定，如果申请人在法定期间或者专利局所指定的期限内未办理相应的手续或者没有提交有关文件，其申请就被视为撤回或者丧失提出某项请求的权利，或者导致有关权利终止的后果。因耽误期限而丧失权利之后，可以在自障碍消除后两个月内，最迟自法定期限或者指定期限届满后两年内或者自收到专利局通知之日起两个月内，请求恢复其权利。

15.9.4 专利权的行使

1. 专利权的归属

根据《中华人民共和国专利法》的规定,执行本单位的任务或者主要是利用本单位的物质条件所完成的职务发明创造,申请专利的权利属于该单位。申请被批准后,专利权归该单位持有(单位为专利权人)。执行本单位的任务所完成的职务发明创造是指:

- 在本职工作中做出的发明创造;
- 履行本单位交付的本职工作之外的任务所做出的发明创造;
- 工作变动(辞职、退休或者调离)后短期内做出的,与其在原单位承担的本职工作或者原单位分配的任务有关的发明创造。

本单位的物质技术条件包括本单位的资金、设备、零部件、原材料或者不对外公开的技术资料等。

非职务发明创造,申请专利的权利属于发明人或者设计人。在中国境内的外资企业和中外合资经营企业的工作人员完成的职务发明创造,申请专利的权利属于该企业,申请被批准后,专利权归申请的企业或者个人所有;两个以上单位协作或者一个单位接受其他单位委托的研究、设计任务所完成的发明创造,除另有协议的以外,申请专利的权利属于完成或者共同完成的单位,申请被批准后,专利权归申请的单位所有或者持有。

2. 专利权人的权利

专利权是一种具有财产权属性的独占权以及由其衍生出来的相应处分权。专利权人的权利包括独占实施权、转让权、实施许可权、放弃权、标记权等。专利权人有缴纳专利年费(也称专利维持费)和实际实施已获专利的发明创造的两项基本义务。

专利权人可通过专利实施许可合同将其依法取得的对某项发明创造的实施权转移给非专利权人行使。任何单位或者个人实施他人专利的,除专利法第十四条规定的以外,都必须与专利权人订立书面实施许可合同,向专利权人支付专利使用费。被许可人无权允许合同规定以外的任何单位或者个人实施该专利。专利实施许可的种类包括独占许可、独家许可、普通许可和部分许可。

15.9.5 专利权的限制

根据《中华人民共和国专利法》的规定,发明专利权的保护期限为自申请日起 20 年,实用新型专利权和外观设计专利权的保护期限为自申请日起 10 年。发明创造专利权的法律效力所及的范围为:

- 发明或者实用新型专利权的保护范围以其权利要求的内容为准,说明书及附图可以用于解释权利要求。

- 外观设计专利权的保护范围以表示在图片或者照片中的该外观设计专利产品为准。

公告授予专利权后,任何单位或个人认为该专利权的授予不符合专利法规定条件的,可以向专利复审委员会提出宣告该专利权无效的请求。专利复审委员会对这种请求进行审查,做出宣告专利权无效或维持专利权的决定。我国专利法规定提出无效宣告请求的时间(启动无效宣告程序的时间)始于“自专利局公告授予专利权之日起”。

专利权因某种法律事实的发生而导致其效力消失的情形称为专利权终止。导致专利权终止的法律事实有:

- 保护期限届满;
- 在专利权保护期限届满前,专利权人以书面形式向专利局声明放弃专利权;
- 在专利权的保护期限内,专利权人没有按照法律的规定交年费。专利权终止日应为上一年度期满日。

专利法允许第三方在某些特殊情况下,可以不经专利权人许可而实施其专利,且其实施行为并不构成侵权。专利权限制的种类包括强制许可、不视为侵犯专利权的行为以及国家计划许可。

15.9.6 专利侵权行为

专利侵权行为是指在专利权的有效期限内,任何单位或者个人在未经专利权人许可,也没有其他法定事由的情况下,擅自以营利为目的实施专利的行为。专利侵权行为主要包括:

- 为生产经营目的制造、使用、销售其专利产品,或者使用其专利方法以及使用、销售依照该专利方法直接获得的产品;
- 为生产经营目的制造、销售其外观设计专利产品;
- 进口依照其专利方法直接获得的产品;
- 产品的包装上标明专利标记和专利号;
- 将非专利产品冒充专利产品或者将非专利方法冒充专利方法等。

对未经专利权人许可,实施其专利的侵权行为,专利权人或者利害关系人可以请求专利管理机关处理。在专利侵权纠纷发生后,专利权人或者利害关系人既可以请求专利管理机关处理,又可以请求人民法院审理。侵犯专利权的诉讼时效为两年,自专利权人或者利害关系人知道或者应当知道侵权行为之日起计算。如果诉讼时效届满,专利权人或者利害关系人不能再请求人民法院保护,同时也不能再向专利管理机关请求保护。

15.10 企业知识产权的保护

知识产权是一种无形的产权,是企业的重要财富,应当把保护知识产权作为现代企业制度的一项基本内容。只有保护好企业拥有的知识产权,才有利于企业保持技术优势,不断发展壮大。

15.10.1 知识产权管理

企业往往会面临软件知识产权的权利归属、软件人才的管理、软件技术的保密以及软件成果的有效利用等一系列问题,这些问题的解决不可能仅仅依靠企业某些领导的行政干预,需要通过企业自身的知识产权管理制度才能解决问题。这样,软件的权利归属不清、人才外流、管理不善而导致软件知识产权泄密和流失的问题,软件的研发及经营活动中不规范的问题,都可以得到有效的解决。

专利权、商标权、著作权等知识产权是企业的经营资源,应建立承担企业知识产权管理的知识产权机构,有效地管理和利用知识产权。参与到软件立项、开发、销售及售后服务的每一环节,把握住企业的技术开发动向以及其他公司的技术开发动向,以知识产权的角度向开发人员、研究人员、管理人员提出建议,并为对外事务提供法律保障。

15.10.2 知识产权的保护和利用

目前,计算机技术和软件技术的知识产权法律保护已形成以《著作权法》保护为主,《著作权法》(包括《计算机软件保护条例》)、《专利法》、《商标法》、《反不正当竞争法》、《合同法》交叉和重叠保护为辅的趋势。例如,源程序及设计文档作为软件的表现形式受著作权法保护,同时作为技术秘密又受《反不正当竞争法》的保护。由于软件具有技术含量高的特点,使得对软件的法律保护成为一种综合性的保护。对于企业来说,仅依靠某项法律或法规不能解决软件的所有知识产权问题。应对企业计算机软件成果的知识产权实施综合性保护,例如,在新技术的开发中重视技术秘密的管理的同时,也应重视专利权的取得。在命名新产品名称时,也应重视商标权的取得,以保护企业的知识产权。企业保护软件成果等知识产权的一般途径如下所示。

1. 明确软件知识产权归属

明确知识产权是归企业还是归制作、设计和开发人员所有,避免企业内部产生归属纠纷。

2. 及时对软件技术秘密采取保密措施

对企业的软件产品或成果中的技术秘密,应当及时采取保密措施,以便把握市场优

势。一旦发生企业“技术秘密”被泄露的情况,则便于认定为技术秘密,以便依法追究泄密行为人的法律责任,保护企业的权益。

3. 依靠专利保护新技术和新产品

专利知识产权是企业重要的经营资源,依靠专利保护企业开发出的新技术或者新产品,使得企业在市场竞争中占据优势。我国采用的是先申请原则,如果有相同技术内容的专利申请,只有最先提出专利申请的企业或者个人才能获得专利权。企业的软件技术或者产品构成专利法律要件的,应当尽早办理申请专利权登记事宜,不能因企业自身的延误,造成企业软件成果新颖性的丧失,从而失去申请专利的时机。

4. 软件产品进入市场之前的商标权和商业秘密保护

企业的软件产品已经冠以商品专用标识或者服务标识,要尽快完成商标或者服务标识的登记注册,保护软件产品的商标专用权。

5. 软件产品进入市场之前进行软件著作权登记

软件著作权登记可以起到公示的作用。软件著作权登记只要求软件的独创性,并不以软件的技术水平作为著作权是否有效的条件,不能等到软件达到某种技术水平后再进行登记,若其他企业或者个人抢先登记,则不利于企业权益的保护。

15.10.3 建立经济约束机制规范调整各种关系

软件企业需要按照经济合同规范各种经济活动,明确权利与义务的关系,建立企业内部以及企业与外部的各种经济约束机制。从目前存在的比较突出的问题来看,软件企业应建立以下各项合同规范。

1. 劳动关系合同

软件企业与企业职工、外聘人员之间应建立合法的劳动关系,同时应就企业的商业秘密(技术秘密和经营秘密)的保密事宜进行约定,建立劳动利益关系合同以及保守企业商业秘密的协议。一些目前不宜马上实行劳动合同的单位,也应通过建立或者健全本单位的有关规章制度的方式进行过渡,以鼓励企业员工的创造性劳动,明确企业开发过程中产生的软件技术成果的归属,以预防企业技术人员流动时造成的技术流失和技术泄密等问题。

2. 软件开发合同

软件企业与外单位合作开发或委托外单位开发软件时,应签订软件权利归属关系等事宜的协议,可按照有关规定签订软件开发合同,约定软件开发各方享有的权利与义务,以及软件技术成果开发完成后的权利归属和经济利益等。如果软件开发方在合作中发现了合同的缺陷,应及早对合同进行补充、完善。

3. 软件许可使用(或者转让)合同

软件企业在经营本企业的软件产品时,应当建立“许可证”(或是转让合同)制度,用软件许可合同(授权书)或者转让合同来明确规定软件使用权的许可(转让)方式、条件、范围、时间等事宜,避免因合同条款的约定不清楚、不明确而导致当事人之间发生扯皮等不愉快的事情,甚至因合同条款无法界定而引发软件侵权纠纷。

第 16 章 标准化基础知识

16.1 标准化的基本概念

16.1.1 标准、标准化的概念

标准(standard)是对重复性事物和概念所做的统一规定。标准以科学、技术和实践经验的综合成果为基础,以获得最佳秩序和促进最佳社会效益为目的,经有关方面协商一致,由主管或公认机构批准,并以规则、指南或特性的文件形式发布,作为共同遵守的准则和依据。规范(specification)、规程(code)都是标准的一种形式。

标准化(standardization)是在经济、技术、科学及管理等社会实践中,以改进产品、过程和服务的适用性,防止贸易壁垒,促进技术合作,促进最大社会效益为目的,对重复性事物和概念通过制定、发布和实施标准,达到统一,获得最佳秩序和社会效益的过程。

16.1.2 标准化的范围和对象

标准化涉及的范围包括生产、经济、技术、科学及管理等社会实践中具有重复性事物和概念以及需要建立统一技术要求的各个领域。在这些领域中,凡具有多次重复使用和需要制定标准的具体产品,以及各种定额、规划、要求、方法、概念等,都可称作标准化的对象。如产品的品种、规格、型式;产品的性能、质量;包装的尺寸、材质、样式;信息表示、信息处理技术;试验方法、测试方法、检验方法、操作方法、抽样方法等各类方法;量值、单位、术语、符号、代码、标志;日常操作、安全操作、劳动保护、安全卫生、保护环境等方面的规范、规程或规定;市场调查、研制开发、物资采购、加工工艺、质量控制、质量检验、销售、售后服务等各阶段的管理事项,等等。

标准化对象一般可分为两大类,一类是标准化的具体对象,即需要制定标准的具体事物;另一类是标准化总体对象,即各种具体对象的全体所构成的整体,通过它可以研究各种具体对象的共同属性、本质和普遍规律。

对一个企业来说,企业的经济活动、技术活动、科研活动和管理活动的全过程及其要素都可作为标准化的范围和对象。企业的活动及其要素具有重复出现的特性,例如,同一产品的反复投入生产;同一检验方法的反复多次进行;同一概念的多次使用;同一类技术活动(如产品设计)同时或相继发生;同一管理事项的重复进行,等等。这些事物和概念的

多次重复活动便产生了按统一标准进行的客观需要和要求,制定标准可以总结以往的经验,选择最佳方案,作为今后实践的目标和依据。如果同一事物和概念在反复多次进行时,没有统一的标准可遵循,就失去了共同的客观依据,企业活动就会因失去准绳而产生混乱。如果不善于利用标准化这个有效工具,每次遇到问题都从头研究、讨论再做出决定,不仅浪费时间和精力,影响效率,而且常常丧失机遇。如果重复进行的活动按标准化实施,遇到同类问题照标准执行,就可避免因人事更动等因素造成同一问题前后处理不一致而带来的问题。这样,领导者、技术人员或管理人员可以把主要精力放在研究和处理企业根本性的、方向性的大问题或新问题上,而由标准化的实施保证企业各项工作的正常进行,使企业活动纳入高效率的轨道。

16.1.3 标准化的实质

标准化的实质是通过制定、发布和实施标准而达到统一。统一的目的是为了保证事物发展所必须的秩序和效率,对事物的形成、功能或其他特性,确定一个适合于一定时期和一定条件的一致性规范,并使这种一致性规范与其取代的对象在功能上达到等效。统一是标准的本质特征,任何标准,都是在一定条件下的“统一规定”。统一的基本含义一般包括以下要点。

(1) 对生产、使用、科研、管理等有关方面进行认真讨论,通过充分协商达成一致的意见,取得共同认可,经过批准后实施。这样制定的标准具有科学性和民主性,具有突出的科学性和民主性的标准,在实施中将具有权威性。

(2) 为了经济而有效地满足需要,对处于自然状态的标准化对象的结构、形式、规格或其他性能进行筛选提炼,剔除其中多余的、低效的、可替换的环节,合理精简并确定出满足要求所必需的高效环节,保持整体构成精简而合理,使其功能和效率(满足全面需要的能力)达到最佳。

(3) 对标准的水平和质量指标等各项内容确定的一致性规范,一般是反映一定时期的水平,因此经统一而确立的一致性仅适用于一定时期和一定条件。随着时间的推移和条件的改变,还须确立新的更高水平的一致性,原先的统一就要由新的统一所代替,以保持标准的先进性和合理性。

(4) 不同层次的标准在不同范围内进行统一,不同类型的标准则是从不同角度(或侧面)进行统一。同一功能的同一对象在同一范围、同一标准化层次上只能选择确定一个统一的规范(包含被取代对象所具备的必要功能),而不能制定出两个或多个规范。

(5) 统一并不意味着绝对统一,即全都统一到只有一种。统一不排斥多样性,有时只限定一个范围,有时则规定几种情况。统一并不限制技术发展,它也有最大自由度原则,例如技术参数的选择要为技术发展留有空间。在同一标准中,统一规定的内容,根据需要

可以是一种或多种;又如产品的分等分级,产品的参数系列,等等。

(6) 标准是科学、技术和实践经验的结晶,其统一的内容和基础是科学、技术和实践经验的综合成果。统一需要建立在科学试验或验证的数据基础上,通过总结、分析并综合国内外有关科学、技术和实践经验的成果以及多年积累的经验,使之科学化和条理化;标准需要吸收新的科研成果、新的技术成就和新的实践经验,以便通过标准的形式去推广新成果、新技术和新经验。

16.1.4 标准化的目的

标准化的目的之一是建立最佳秩序,即建立一定环境和一定条件下的最合理的秩序。通过标准化在社会生产组成部分之间进行协调,确立共同遵循的准则,建立稳定和最佳的生产、技术、安全、管理等秩序,使生产活动和经营管理活动井然有序,避免混乱,提高效率。标准化的另一目的就是获得最佳效益。一定范围的标准,是按一定范围内的技术效益和经济效果的目标制定出来的,它不仅考虑了标准在技术上的先进性,还考虑到经济上的合理性以及企业的最佳经济效益。标准虽然不是商品,但却能加速商品的生产和流通,能极其显著地提高劳动生产率和资源的转化效率,实现商品生产的合理化、高效率 and 低成本,给企业带来可观的利润。一些工业发达国家把标准化作为企业经营管理、获取利润以及进行竞争的法宝和秘密武器。特别是一些著名公司,往往都建立标准化体系,以保证其利润和竞争目标的实现。随着商品经济的发展和市场的扩大,特别是专业化生产和国际协作的出现,为标准的应用创造了广阔的空间,并为一体化的世界经济准备了一套保证公平贸易的准则。

16.2 标准化过程模式

标准是标准化活动的产物,其目的和作用都是通过制定和贯彻具体的标准来体现的。标准化不是一个孤立的事物,而是一个活动过程。一个过程可分为几个子过程,其最后一个子过程,是一个过程的终结,也是下一个过程的开始。通过总结前一个过程的经验和教训,并依据客观环境的新变化和新要求,提出标准修订的新目标。这是一个不断循环,螺旋式上升的运动过程,每完成一个循环,标准的水平将提高一步。

标准化活动过程一般包括标准产生(调查、研究、形成草案、批准发布)子过程、标准实施(宣传、普及、监督、咨询)子过程和标准更新(复审、废止或修订)等子过程。

16.2.1 标准的制定

制定标准的过程,实际上就是总结和积累人类社会实践经验的过程,每一个新标准的

产生,都标志着某一领域或某项活动的经验被规范化。标准的产生一般包括调查研究、制定计划(立项)、起草标准、征求意见、审查、批准发布等标准生成阶段。这个过程中产生的问题,有程序性的问题(如征求意见、审查等);更有实质性的问题,即标准的适用性、可行性、先进性方面的问题。这个过程随标准化对象的不同而不同。起草标准时一般需要吸收新的科研成果、新的技术成就和新的实践经验,以便通过标准的形式去推广新成果、新技术和新经验。为保证标准的编写质量,便于资料管理,体现其严肃性,标准文件有统一的格式。标准从制定到批准发布的一整套工作程序和审批制度,体现了标准产生的科学规律,表现出其法律特性。ISO 和 IEC 是两个国际标准化组织,为规范国际标准的产生过程发布了指导性文件。我国依据该指导性文件制定了相应的国家标准,以规范国家标准的制定程序。

16.2.2 标准的实施

标准的实施过程,实际上就是推广和普及已被规范化的实践经验的过程。标准的实施过程包括哪些活动内容并没有统一的规定,但一般包括标准的宣传、贯彻执行和监督检查等。标准化活动是一项有组织的活动,并且始终是一个组织的行为。通常,国家、区域和行业的标准化管理组织以及标准化团体,通过宏观管理(标准规划、计划和协调),健全以国际标准、区域性标准、国家标准以及行业标准为主的标准文献资料,建立健全行业和企业标准备案管理制度,提高生产、经销和服务单位执行标准的自觉性。我国强制性标准的实施通过强制性的监督检查来推动,依法开展标准的实施与监督。对于推荐性标准的实施,尚无明确有效的措施,需要建立和完善社会主义市场经济体制下的技术法规。《标准化法》、《国家标准管理办法》等法规规定了我国标准化工作的方针、政策、任务和标准化体制等。它们是我国推行标准化,实施标准化管理和监督的重要依据。企业根据企业和市场的需求及技术发展趋势的需要,开展标准立项的前期研究,为标准制定提供可靠的依据。开展标准化审查,制止不符合强制性标准规定、不符合标准化技术政策的技术和产品的研制和推广;确定重点标准的实施与监督的项目计划,对各个环节进行全面动态跟踪管理。在工业发达国家,标准一般是推荐性的,其实施的推动力来自标准本身的科学性所产生的信任,以及通过产品认证所产生的权威性。

16.2.3 标准的更新

已经实现了标准化的事物,实施一段时间后,有可能突破原先的规定。如果有新的需求,使某些环节的标准失去意义,就需要修订或再制定标准。标准的更新是实践经验的深化和提高的过程。通过信息反馈总结经验,找出问题,依据客观环境的新变化和新要求,提出标准修订的新目标,更新标准。人类社会实践是一个永不止息的活动,标准化也是一

个永无止境的过程。

1. 标准复审

已经发布实施的现有标准(包括已确认或修改补充的标准),经过一段时间后,需要对其内容进行再次审查,以确保其有效性、先进性和适用性。自标准实施之日起,至标准复审重新确认、修订或废止的时间,称为标准的有效期(也称为标龄)。由于各个国家的情况不同,标准有效期也不同。例如,ISO标准每5年复审一次,平均标龄为4.92年;1988年发布的《中华人民共和国标准化法实施条例》中规定,标准实施后的复审周期一般不超过5年,即我国的国家标准有效期一般为5年。标准复审工作由各主管部门或标准化技术委员会组织进行,对需要复审的标准要收集实施中的问题并进行分类整理。复审时可采用会议审查或函审,经复审的标准要用书面形式报告复审结果,如复审简况、处理意见、复审结论和依据等,以确认现行标准是否继续有效,是否需要修订或者废止,确保标准的时效性。

2. 标准确认

经复审后的标准,若其内容仍符合当前科学技术水平并适合经济建设的需要,无须修改或只须作编辑性修改,可确认为继续有效。经确认的标准,发布时不改变标准的顺序号和年号。标准再版时在标准封面写明××××年确认字样。

经过复审后的标准,若其内容须完善和补充,只要对标准条文、图、表做少量修改补充,仍可继续使用的,则采用标准修改单的形式,对需要修改补充的内容予以完善后,按照标准的制定程序,经标准化主管单位批准发布。

3. 标准修订

经复审后的标准,若其内容需要做较大修改才能适应生产和使用的需要以及科学技术发展的需要,则应作为修订项目。标准修订的程序按制定标准的程序执行。修订后的标准顺序号不变,年号改为重新修订发布时的年号。

16.3 标准的分类

标准化工作是一项复杂的系统工程,标准为适应不同的要求构成了一个庞大而复杂的系统。为便于研究和应用,可以从不同的角度和属性将标准进行分类。

16.3.1 根据适用范围分类

根据标准制定的机构和标准适用的范围,可分为国际标准、国家标准、行业标准、企业(机构)标准及项目(课题)标准。

1. 国际标准(international standard)

国际标准是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准,以及ISO出版的《国际标准题内关键词索引(KWIC Index)》中收录的其他国际组织制定的标准。国际标准在世界范围内使用,各国可以自愿采用,不强制使用。但由于国际标准集中了一些先进工业国家的技术经验,各个国家为外贸上和其他本国利益考虑往往都积极采用国际标准。

2. 国家标准(national standard)

国家标准是由政府或国家级的机构制定或批准的、适用于全国范围的标准,是一个国家标准体系的主体和基础,国内各级标准必须服从且不得与之相抵触。常见的国家标准有:

(1) 中华人民共和国国家标准(GB)是我国最高标准化机构中华人民共和国国家技术监督局所公布实施的标准,简称为“国标”。

(2) 美国国家标准是美国国家标准协会(American National Standards institute, ANSI)制定的标准。

(3) 英国国家标准(British Standard, BS)是英国标准学会(BSI)制定的标准。

(4) 日本工业标准(Japanese Industrial Standard, JIS)是日本工业标准调查会(JISC)制定的标准。

3. 区域标准(regional standard)

区域标准(也称地区标准)泛指世界上按地理、经济或政治划分的某一区域标准化团体所通过的标准。它是为了某一区域的利益而建立的标准。通常,地区标准主要是指太平洋地区标准会议(PASC)、欧洲标准化委员会(CEN)、亚洲标准咨询委员会(ASAC)、非洲地区标准化组织(ARSO)等地区组织所制定和使用的标准。

4. 行业标准(specialized standard)

由行业机构、学术团体或国防机构制定,并适用于某个业务领域的标准。

(1) 美国电气和电子工程师学会标准 IEEE。IEEE 通过的标准常常要报请 ANSI 审批,使其具有国家标准的性质。因此,IEEE 公布的标准常冠有 ANSI 字头。例如,ANSI/IEEE Str 828—1983(软件配置管理计划标准)。

(2) 中华人民共和国国家军用标准 GJB。这是由我国国防科学技术工业委员会批准,适用于国防部门和军队使用的标准。例如,1988 年发布实施的 GJB473—88(军用软件开发规范)。

(3) 美国国防部标准 DOD-STD(Department of Defense-Standards)。适用于美国国防部门。美国军用标准 MIL-S(Military-Standards),适用于美军内部。

5. 企业标准(company standard)

由企业或公司批准、发布的标准。某些产品标准由其上级主管机构批准、发布。例如,美国 IBM 公司通用产品部(General Products Division)1984 年制定的“程序设计开发指南”,仅供该公司内部使用。

6. 项目规范(project specification)

由某一科研生产项目组织制定,且为该项任务专用的软件工程规范。例如,计算机集成制造系统(CIMS)的软件工程规范。

7. 我国的标准分类

根据《中华人民共和国标准化法》的规定,我国标准分为国家标准、行业标准、地方标准和企业标准等 4 类。这 4 类标准主要是适用的范围不同,不是标准技术水平高低的分级。

(1) 国家标准:由国务院标准化行政主管部门制定的,需要在全国范围内统一的技术要求,称为国家标准。

(2) 行业标准:没有国家标准而又须在全国某个行业范围内统一的技术标准,由国务院有关行政主管部门制定并报国务院标准化行政主管部门备案的标准,称为行业标准。

(3) 地方标准:没有国家标准和行业标准而又须在省、自治区、直辖市范围内统一的要求,由省、自治区、直辖市标准化行政主管部门制定并报国务院标准化行政主管部门和国务院有关行业行政主管部门备案的标准,称为地方标准。

(4) 企业标准:企业生产的产品没有国家标准、行业标准和地方标准,由企业自行组织制定、作为组织生产依据的相应标准,或者在企业内制定的、比国家标准、行业标准或地方标准更严格的企业标准,并按省、自治区、直辖市人民政府的规定备案的标准(不含内控标准),称为企业标准。

16.3.2 根据标准的性质分类

根据标准的性质可分为技术标准、管理标准和工作标准。

1. 技术标准(technique standard)

技术标准是针对重复性的技术事项而制定的标准,是从事生产、建设及商品流通时需要共同遵守的一种技术依据。按其标准化对象特征和作用,可分为基础标准、产品标准、方法标准、安全卫生与环境保护标准等;按其标准化对象在生产流程中的作用,可分为零部件标准、工装标准、设备维修保养标准及检查标准等;按标准的强制程度,可分为强制性与推荐性标准;按标准在企业中的适用范围,又可分为公司标准、工厂标准和科室标准等。

2. 管理标准(administrative standard)

管理标准是管理机构为行使其管理职能而制定的具有特定管理功能的标准,主要用

于规定人们在生产活动和社会实践中的组织结构、职责权限、过程方法、程序文件、资源分配以及方针、目标、措施、影响管理的因素等事宜,是合理组织国民经济,正确处理各种生产关系,正确实现合理分配,提高生产效率和效益的依据。在实际工作中通常按照标准所起的作用不同,将管理标准分为技术管理标准、生产组织标准、经济管理标准、行政管理标准和业务管理标准等。

3. 工作标准(work standard)

为协调整个工作过程,提高工作质量和效率,针对具体岗位的工作制定的标准。对工作的内容、方法、程序和质量要求所制定的标准,称为工作标准。工作标准的内容包括各岗位的职责和任务、每项任务的数量、质量要求及完成期限、完成各项任务的程序和方法、与相关岗位的协调、信息传递方式以及工作人员的考核与奖罚方法等。对生产和业务处理的先后顺序、内容和要达到的要求所作的规定称为工作程序标准。工作程序标准是工作标准的一种,其目的是使各项工作条理化、标准化和规范化,以求得最佳工作秩序、工作质量和工作效率。以管理工作为对象所制定的标准,称为管理工作标准。管理工作标准的内容主要包括工作范围、内容和要求、与相关工作的关系、工作条件、工作人员的职权与必备条件、工作人员的考核、评价及奖惩办法等。

16.3.3 根据标准化的对象和作用分类

根据标准的对象和作用,标准可分为基础标准、产品标准、方法标准、安全标准、卫生标准、环境保护标准、服务标准等。

1. 基础标准(basic standard)

在一定范围内作为其他标准的基础并普遍适用,具有广泛指导意义的标准。如名词、术语、符号、代号、标识、方法、模数、公差与配合、基本参数系列、产品系列型号、产品环境条件、可靠性要求等。在某领域中,基础标准通常是覆盖面最大的标准,也是该领域中所有标准的共同基础。

2. 产品标准(product standard)

为保证产品的适用性,对产品必须达到的某些或全部特性要求所制定的标准。产品标准是一定时期和一定范围内具有约束力的产品技术准则,是产品生产、质量检验、选购验收、使用维护和洽谈贸易的技术依据。产品标准的主要内容包括产品的适用范围、产品的品种、规格和结构形式、产品的主要性能、产品的试验、检验方法和验收规则以及产品的包装、储存和运输等方面的要求。产品标准可分为完全的产品标准和不完全的产品标准(品种标准):完全的产品标准是规定产品术语、符号、技术性能、试验方法、检验、包装、储运等全部要求的标准;不完全的产品标准中只规定品种、规格、类型、样式与尺寸、参数系列等。

3. 方法标准(method standard)

以各种方法为对象而制定的标准。方法标准一般包括两类：一类是以试验、检查、分析、抽样、统计、计算、测定、作业等方法为对象制定的标准。例如，试验方法、检查方法、分析方法、测定方法、抽样方法、设计规范、计算方法、工艺规程、作业指导书、生产方法、操作方法及包装、运输方法等。另一类是为合理生产优质产品，并在生产、作业、试验、业务处理等方面为提高效率而制定的标准。

4. 安全标准(safety standard)

以保护人的安全或物品的安全为对象和目的而制定的标准。安全标准一般有两种形式：一种为专门目的的安全标准；另一种是在产品标准或工艺标准中列出有关安全的要求和指标。例如，安全标准可包括劳动安全标准、锅炉和压力容器安全标准、电气安全标准和消费品安全标准等。安全标准一般均为强制性标准，由国家通过法律或法令形式强制执行。

5. 卫生标准(hygienic standard)

为保护人的健康，对食品、医药及其他方面的卫生要求而制定的标准。卫生标准是专门以卫生(工业卫生、劳动卫生等)要求为对象和目的制定的标准，如食品卫生标准，规定了食品中有害物质或病菌的限量。

6. 环境保护标准(environment protection standard)

为保护环境不受污染和有利于生态平衡，对大气、水体、土壤、噪声、振动、电磁波等环境质量、污染管理、监测方法及其他事项制定的标准。例如水质标准、噪声标准等。

7. 服务标准(service standard)

为提高服务质量，使某项服务工作达到要求所制定的标准。服务包括技术服务，即产品售后服务；也包括第三产业，即旅游、商业、交通运输、金融、电讯、信息、保险、医疗等服务行业。

16.3.4 根据法律的约束性分类

根据标准的法律约束性，可分为强制性标准和推荐性标准。

1. 强制性标准

根据《中华人民共和国标准化法》的规定，企业和有关部门对涉及其经营、生产、服务、管理有关的强制性标准都必须严格执行，任何单位和个人不得擅自更改或降低标准。对违反强制性标准而造成不良后果甚至重大事故者，由法律、行政法规规定的行政主管部门依法根据情节轻重给予行政处罚，直至由司法机关追究刑事责任。

强制性标准是国家技术法规的重要组成部分，它符合世界贸易组织贸易技术壁垒协定关于“技术法规”的定义，即“强制执行的产品特性或相应加工方法的规定包括可适用的

行政管理规定在内的文件。技术法规也可包括(或专门规定)用于产品、加工或生产方法的术语、符号、包装标志或标签要求”,为使我国强制性标准与 WTO/TBT 规定衔接,其范围限制在国家安全、防止欺诈行为、保护人身健康与安全、保护动物植物的生命和健康以及保护环境等方面。

2. 推荐性标准

在生产、交换、使用等方面,通过经济手段或市场调节而自愿采用的一类标准称为推荐性标准。这类标准不具有强制性,任何单位均有权决定是否采用。违背这类标准,不构成经济或法律方面的责任。应当指出的是,推荐性标准一经接受并采用,或由各方商定后纳入经济合同中,就成为各方必须共同遵守的技术依据,具有法律上的约束性。

推荐性标准是指导性标准,是由公认机构批准的,非强制性的,为了通用或反复使用的目的,为产品或相关生产方法提供规则、指南或特性的文件。标准也可以包括(或专门规定)用于产品、加工或生产方法的术语、符号、包装标准或标签要求。由于推荐性标准是协调一致的文件,不受政府和社会团体的利益干预,能更科学地规定特性或指导生产,故《标准化法》鼓励企业积极采用推荐性标准。为了防止企业利用标准欺诈消费者,要求采用低于推荐性标准的企业标准组织生产的企业向消费者明示其产品标准类型。

16.4 标准的代号和编号

1. ISO 的代号和编号

ISO 代号和编号的格式为: ISO+标准号+[杠+分标准号]+冒号+发布年号(方括号中的内容可有可无)。例如,“ISO8402: 1987 和 ISO9000-1: 1994”是 ISO 标准的代号和编号。

2. 国家标准的代号和编号

我国国家标准的代号由大写汉字拼音字母构成。强制性国家标准代号为 GB,推荐性国家标准的代号为 GB/T。

国家标准的编号由国家标准的代号、标准发布顺序号和标准发布年代号(4 位数)组成。

(1) 强制性国家标准: GB ×××××.××××。

(2) 推荐性国家标准: GB/T ××××× — ××××。

3. 行业标准的代号和编号

(1) 行业标准代号: 行业标准代号由汉字拼音大写字母组成。由国务院各有关行政主管部门提出其所管理的行业标准范围的申请报告,国务院标准化行政主管部门审查确定并正式公布该行业标准代号。已正式公布的行业代号有 QJ(航天)、SJ(电子)、JB(机

械)、JR(金融系统)等。

(2) 行业标准的编号：行业标准的编号由行业标准代号、标准发布顺序及标准发布年代号(4 位数)组成。表示方法如下所示。

- 强制性行业标准编号：×× ×××× — ××××。
- 推荐性行业标准编号：××/T ×××× — ××××。

4. 地方标准的代号和编号

(1) 地方标准的代号：地方标准代号由大写汉字拼音 DB 加上省、自治区、直辖市行政区划代码的前两位数字(如北京市 11、天津市 12、上海市 31 等)，再加上斜线 T 组成推荐性地方标准，不加“/T”为强制性地方标准。表示方法如下所示。

- 强制性地方标准：DB××。
- 推荐性地方标准：DB××/T。

(2) 地方标准的编号：地方标准的编号由地方标准代号、地方标准发布顺序号、标准发布年代号(4 位数)3 部分组成。表示方法如下所示。

- 强制性地方标准：DB×× ××× — ××××。
- 推荐性地方标准：DB××/T ××× — ××××。

5. 企业标准的代号和编号

(1) 企业标准的代号：企业标准的代号由汉字大写拼音字母 Q 加斜线再加企业代号组成，企业代号可用大写拼音字母或阿拉数字或两者兼用所组成。企业代号按中央所属企业和地方企业分别由国务院有关行政主管部门或省、自治区、直辖市政府标准化行政主管部门会同同级有关行政主管部门加以规定。例如“Q/×××”。企业标准一经制定颁布，即对整个企业具有约束性，是企业法规性文件，没有强制性企业标准和推荐企业标准之分。

(2) 企业标准的编号：企业标准的编号由企业标准代号、标准发布顺序号和标准发布年代号(4 位数)组成，表示方法为“Q/××× ×××× — ××××”。

16.5 国际标准和国外先进标准

由于国际贸易广泛开展，产品在国际市场上的竞争越来越激烈，要求产品不但要具有高的质量，好的性能，还要具有广泛的通用性、互换性。这就需要采用统一的、先进的标准，按照国际上统一的标准生产，避免采用的标准不一致，带来国际贸易障碍。国际标准和国外先进标准集中了一些先进工业国家的技术经验，加之各国考虑外贸上的利益，世界各国都积极采用国际标准或先进的标准。许多国家直接把国际标准作为本国标准使用。采用国际标准和国外先进标准是我国一项重大技术经济政策，是促进技术进步，提高产品

质量,扩大对外开放,加快与国际惯例接轨,发展社会主义市场经济的一项重要措施。

16.5.1 国际标准

国际标准是指国际标准化组织(ISO)、国际电工委员会(IEC)所制定的标准,以及ISO出版的《国际标准上下文内关键词索引(KWIC Index)》中收录的其他国际组织制定的标准。1983年3月出版的KWIC索引(第一版)中共收录了24个国际组织制定的7600个标准,其中ISO标准占68%,IEC标准占18.5%,其他22个国际组织的标准共968个,占13.5%。1989年的KWIC索引(第二版)共收录了ISO与IEC制定的800个标准,以及其他27个国际组织的1200多条标准。ISO推荐列入KWIC索引的有27个国际组织及一些未列入《KWIC Index》的国际组织所制定的某些被国际公认的标准。这27个国际组织制定的标准化文献主要有国际标准、国际建议、国际公约、国际公约的技术附录和国际代码,也有经各国政府认可的强制性要求,对国际贸易业务服务和信息交流具有重要影响。

16.5.2 国外先进标准

国外先进标准是指国际上有权威的区域性标准,世界上经济发达国家的国家标准和通行的团体标准,包括知名企业标准在内的其他国际上公认的先进标准。

(1) 有国际权威的区域性标准。如欧洲标准化委员会(CEN)、欧洲电工标准化委员会(CENELEC)、欧洲广播联盟(EBU)、亚洲大洋洲开放系统互连研讨会(AOW)、亚洲电子数据交换理事会(ASEB)等制定的标准。

(2) 世界经济技术发达国家的国家标准。如美国国家标准(ANSI)、德国国家标准(DIN)、英国国家标准(BS)、日本国工业标准(JIS)、瑞典国家标准(SIS)、法国国家标准(NF)、瑞士国家标准(SNV)、意大利国家标准(UNI)、俄罗斯国家标准(ТОСТ)等。

(3) 国际公认的行业性团体标准。如美国材料与实验协会标准(ASTM)、美国石油学会标准(API)、美国军用标准(MIL)、美国电气制造商协会标准(NEMA)、美国电影电视工程师协会标准(SMPTE)、美国机械工程师协会标准(ASME)、英国石油学会(IP)等。

(4) 国际公认的先进企业标准。如美国IBM公司、美国HP公司、芬兰诺基亚公司、瑞士钟表公司等企业标准等。

16.5.3 采用国际标准和国外先进标准

采用是指采纳和应用。采用国际标准和国外先进标准是把国际标准和国外先进标准或其内容,通过分析研究,不同程度地订入(编入)我国标准,并贯彻执行。将国际标准和国外先进标准纳入国家标准的主要方法如下所述。

(1) 认可法：由国家标准机构直接宣布某项国际标准为国家标准，其具体办法是发布认可公告或通知，公告和通知中一般不附带国际标准的正文，也不在原标准文本上加注采用国家的编号。

(2) 封面法：在国际标准上加上采用国国家标准的编号，并附简要说明和要求，如说明对原标准做了哪些编辑修改，以及如何贯彻等要求。

(3) 完全重印法：将国际标准翻译或不作翻译，采用原标准标题，重新印刷作为国家标准，并可在国际标准正文前面，加一篇引言，做一些说明或解释、要求。

(4) 翻译法：国家标准采用国际标准的译文，可以用两种文字（原文和译文）或一种文字出版，采用时也可在前言中说明对采用的国际标准做了哪些编辑性修改，或做一些要求和说明。

(5) 重新制定法：根据某项国际标准，重新起草国家标准，即把国际标准融合到国家标准之中，或做层次上的修改或做结构上的变动，但一般要保留国际标准的主要指标，或基本上保留原结构格局。

(6) 包括与引用法：制定国家标准时，完全引用或部分引用国际标准的内容。根据国际标准的“包容”情况及专业深度，可包括国际标准的一部分，其余根据需要补充新的内容和指标。可以选择相关部分进行贯彻，其余部分不贯彻。

16.5.4 采用程度的概念

采用国际标准或国外先进标准的程度，分为等同采用、等效采用和非等效采用。

(1) 等同采用：指国家标准等同于国际标准，仅有或没有编辑性修改。编辑性修改是指不改变标准的技术内容的修改。如纠正排版或印刷错误，改变标点符号，增加不改变技术内容的说明和指示等。因此，可以认为等同采用就是指国家标准与国际标准相同，不做或稍做编辑性修改，编写方法完全对应。

(2) 等效采用：指国家标准等效于国际标准，技术内容上只有很小差异，编辑上不完全相同，编写方法不完全对应。如奥地利内河船舶噪声测量标准(ONORMS 5022)中包括一份试验报告的推荐格式，而相应的国际标准 ISO 2922 中没有此内容。

(3) 非等效采用：指国家标准不等效于国际标准，在技术上有重大技术差异。即国家标准中有国际标准不能接受的条款，或者在国际标准中有国家标准不能接受的条款。在技术上有重大差异的情况下，虽然国家标准制定以国际标准为基础，并在很大程度上与国际标准相适应，但不能使用“等效”这个术语。通常包括以下 3 种情况。

- 国家标准包含的内容比国际标准少：国家标准较国际要求低或仅选国际标准的部分内容。国家标准与国际标准之间没有互相接受条款的“逆定理”情况。
- 国家标准包含的内容比国际标准多：国家标准增加了内容或类型，且具有较高要

求等。也没有“逆定理”情况。

- 国家标准与国际标准有重叠：部分内容完全相同或技术上相同，但在其他内容上却互不包括对方的内容。

采用国际标准或国外先进标准，按国家标准 GB161 的规定编写。采用程度符号用缩写字母表示，等同采用为 idt 或 IDT，等效采用为 eqv 或 EQV，非等效采用为 neq 或 NEQ。

- 等同采用：GB ××××—××××(idt ISO ××××—××××)。
- 等效采用：GB ××××—××××(eqv ISO ××××—××××)。
- 非等效采用：GB ××××—××××(neq ISO××××—××××)。

16.5.5 采用国际标准和国外先进标准的原则

采用国际标准和国外先进标准，通常遵循的原则是：

(1) 根据我国国民经济发展的需要，确定一定时期采用国际标准和国外先进标准的方向及任务。国民经济发展的需要是采用国际标准和国外先进标准的出发点。当国民经济处于建立社会主义经济体系初期，采用国际标准和国外先进标准就是要从战略上、从国家长远利益上考虑，突出国际标准中的重大基础标准和通用方法标准的采用问题。当国民经济发展到一定阶段，如产品质量要赶超世界先进水平时，对国际标准和国外先进标准中的先进产品标准和质量标准就成为采用的重要对象。

(2) 很多国际标准是经多年实际验证后公认的，通常不必都去进行实践验证。为加快采用国际标准和国外先进的速度，一般是简化制定手续，基本上采取“先拿来用，然后实践验证，再补充修改”的模式。

(3) 促进产品质量与水平的提高是当前采用国际标准和国外先进标准的一项重要原则。产品质量问题首先有标准问题，只有采用了先进的国际标准或先进的国外标准，才能提高我国的标准水平。只有提高了标准水平，才能有力地促进产品质量的提高。要赶超世界先进水平就要采用国际标准和国外先进标准。

(4) 要紧密结合我国实际情况、自然资源和自然条件，符合国家的有关法令、法规和政策，做到技术先进，经济合理，安全可靠，使用方便，促进生产力发展。

(5) 对于国际标准中的基础标准、方法标准、原材料标准和通用零部件标准，需要先行采用。通过的基础标准、方法标准以及有关安全、卫生、环境保护等方面的标准，一般应与国际标准协调一致。

(6) 在技术引进和设备进口中采用国际标准，应符合《技术引进和设备进口标准化审查管理办法(试行)》中的规定。例如，原则上不引进和进口英制设备，等等。

(7) 当国际标准不能满足要求，或尚无国际标准时，应参照上述原则，积极采用国外

先进标准。

16.6 信息技术标准化

信息技术标准化是围绕信息技术开发、信息产品研制和信息系统建设、运行与管理而开展的一系列标准化工作,主要包括信息技术术语、信息表示、汉字信息处理技术、媒体、软件工程、数据库、网络通信、电子数据交换、电子卡、管理信息系统、计算机辅助技术等方面的标准化。

16.6.1 信息编码标准化

编码是一种信息表现形式。在一定条件下,它对事物或概念的描述,比自然语言要直接、简洁、准确和有力。例如,交通管理人员使用的不同颜色的信号灯,就是一种特定的信息代码,用来表示行驶、慢驶、停止等交通指令信息十分方便准确。编码作为一种形式,应具有-致性,在一定范围内为人们共同接受,否则就不能发挥作用。例如,邮政编码必须在一个国家范围内一致;公共汽车交通路线编码,必须在一个城市或地区范围内一致;学生学号编码必须在学校系统内一致。要保证信息编码的一致性,就要对编码对象的确定、对象特性的选择、编码方法和代码设计进行标准化。

编码是一种信息交换的技术手段。对信息进行编码实际上是对文字、音频、图形、图像等信息进行处理,使之量化,从而便于利用各种通信设备进行信息传递和利用计算机进行信息处理。作为一种信息交换的技术手段,必须保证信息交换的一致性。例如,计算机内部的所有数据都是用二进制数表示的,但人们向计算机输入的信息,则是人类语言中的数字、文字和专用符号,经计算机处理后的输出也必须是人们所能识别的字符。每个字符所对应的二进制数,便是该字符的代码。各计算机所定义的输入输出符号集和每个符号的代码,便是该计算机的编码系统。只有具有相同编码系统的计算机,才可能接受用户编写的同一符号的程序。为了统一编码系统,人们借助了标准化这个工具,制定了各种标准代码,如国际上比较通用的 ASCII 码(美国信息交换标准代码)。

16.6.2 条码标准化

条码是一种特殊的代码。国家标准《GB12950—91 条码系统通用术语 条码符号术语》中定义,条码是“一组规则排列的条、空及其对应字符组成的标记,用以表示一定的信息”。条码中的条、空分别由两种不同深浅的颜色(通常为黑、白色)表示,并满足一定的光学对比度要求,其目的是便于光电扫描设备识读后将数据输入计算机。条码中的字符供人们直接识读,或通过键盘向计算机输入数据。目前国际上广泛使用的条码是国际物品

编码协会的标准化条码 EAN。我国于 1991 年 4 月正式加入国际物品编码协会,我国国家标准 GB904—91 通用商品条码的结构与 EAN 条码结构相同,由 13 位数字码以及对应的条码组成,即前缀码(3 位)、制造厂商代码(4 位)、商品代码(5 位)、检验码(1 位)。3 位前缀码是标识国家或地区的代码。我国的国家代码为 690。

16.6.3 汉字编码标准化

汉字编码是对每一个汉字按一定的规律用若干个字母、数字和符号表示出来。汉字编码的方法很多,主要有数字编码,如电报码和四角号码字典中的号码都是用数字对汉字进行编码;拼音编码,即用汉字的拼音字母对汉字进行编码;字形编码,即用汉字的偏旁部首和笔画结构与各个英文字母或符号相对应,再加上一些组合规则来表示相应的汉字。对每一种汉字编码,计算机内部都有一种相应的二进制内部码,不同的汉字编码,在使用上不能替换。若把各种编码方案都存入计算机供人们选择,在技术上和使用效果上则是困难的和不经济的。我国在汉字编码标准化方面取得的突出成就就是信息交换用汉字编码字符集国家标准的制定。该字符集共有 6 集。其中,GB2312—80 信息交换用汉字编码字符集是基本集,收入常用基本汉字和字符 7445 个。GB7589—87 和 GB7590—87 分别是第二辅助集和第四辅助集,各收入现代规范汉字 7426 个。GB/T12345—90 是辅助集,它与第三辅助集和第五辅助集分别是与基本集、第二辅助集和第四辅助集相对应的繁体字的汉字字符集。除汉字编码标准化外,汉字信息处理标准化的内容还包括汉字键盘输入的标准化、汉字文字识别输入和语音识别输入的标准化、汉字输出字体和质量的标准以及汉字属性和汉语词语的标准化等。

16.6.4 软件工程标准化

随着软件工程学科的发展,人们对计算机软件的认识逐渐深入。软件工作的范围从只是使用程序设计语言编写程序,扩展到整个软件生存期。软件工程的目的是改善软件开发的组织,降低开发成本,缩短开发时间,提高工作效率,提高软件质量。它在内容上包括软件开发的软件概念形成、需求分析、计划组织、系统分析与设计、结构程序设计、软件调试、软件测试和验收、安装和检验、软件运行和维护,以及软件运行的终止和引退(为新的软件所代替)。同时还有许多技术管理工作,如过程管理、产品管理、资源管理,以及确认与验证工作,如评审、审计与产品分析等。软件工程最显著的特点就是把个别的、自发的、分散的、手工的软件开发变成一种社会化的软件生产方式。软件生产的社会化必然要求软件工程实行标准化。软件工程标准的类型也是多方面的,常常是跨越软件生存期各个阶段。所有这些方面都应逐步建立标准或规范。软件工程标准化的主要内容包括过程标准(如方法、技术、量度等)、产品标准(如需求、设计、部件、描述、计划、报告等)、专业标

准(如道德准则、认证等)、记法标准(如术语、表示法、语言等)、开发规范(准则、方法、规程等)、文件规范(文件范围、文件编制、文件内容要求、编写提示)、维护规范(软件维护、组织与实施等)以及质量规范(软件质量保证、软件配置管理、软件测试、软件验收等)等。

我国 1983 年 5 月成立“计算机与信息处理标准化技术委员会”,下设 13 个分技术委员会,其中程序设计语言分技术委员会和软件工程分技术委员会与软件相关。我国推行软件工程标准化工作的总原则是向国际标准靠拢,对于能够在我国适用的标准全部实行等同采用,以促进国际交流。虽然我国的软件工程标准化工作仍处于起步阶段,但在提高我国软件工程水平,促进软件产业的发展以及加强与国外的软件交流等方面必将起到应有的作用。现已得到国家批准的软件工程国家标准如下所示。

1. 基础标准

- (1) 信息处理—程序构造及其表示法的约定 GB/T 13502—92。
- (2) 信息处理系统-计算机系统配置图符号及其约定 GB/T 14085—93。
- (3) 软件工程技术术语标准 GB/T 11457—89。
- (4) 软件工程标准分类法 GB/T 15538—95。

2. 开发标准

- (5) 软件开发规范 GB 8566—88。
- (6) 计算机软件单元测试 GB/T 15532—95。
- (7) 软件维护指南 GB/T 14079—93。

3. 文档标准

- (8) 计算机软件产品开发文件编制指南 GB 8567—88。
- (9) 计算机软件需求说明编制指南 GB/T 9385—88。
- (10) 计算机软件测试文件编制指南 GB/T 9386—88。

4. 管理标准

- (11) 计算机软件配置管理计划规范 GB/T 12505—90。
- (12) 计算机软件质量保证计划规范 GB/T 12504—90。
- (13) 计算机软件可靠性和可维护性管理 GB/T 14394—93。
- (14) 信息技术、软件产品评价、质量特性及其使用指南 GB/T 16260—96。

16.7 标准化组织

16.7.1 国际标准化组织

ISO 和 IEC 是世界上两个最大、最具有权威的国际标准化组织。目前,由 ISO 确认

并公布的国际标准化组织还有国际计量局(BIPM)、联合国教科文组织(UNESCO)、世界卫生组织(WHO)、世界知识产权组织(WIPO)、国际信息与文献联合会(FID)、国际法定计量组织(OIML)等 27 个国际组织。

1. 国际标准化组织(International Organization for Standardization, ISO)

国际标准化组织是世界上最大的非政府性的,由各国标准化团体(ISO 成员团体)组成的世界性联合专门机构。它成立于 1947 年 2 月,其宗旨是在世界范围内促进标准化工作的发展,以利于国际资源的交流和合理配置,扩大各国在知识、科学、技术和经济领域的合作;其主要活动是制定国际标准,协调世界范围内的标准化工作,组织各成员国和技术委员会进行交流,以及与其他国际性组织进行合作,共同研究有关标准问题,出版 ISO 国际标准。制定国际标准的工作通常由 ISO 的技术委员会完成,各成员团体若对某技术委员会确立的项目感兴趣,均有权参加该委员会的工作。与 ISO 保持联系的各国际组织(官方的或非官方的)也可参加有关工作。此外,ISO 还负责协调世界范围内的标准化工作,组织各成员国和技术委员会进行情报交流,并和其他国际性组织保持联系和合作,共同研究感兴趣的有关标准化问题。在电工技术标准化方面,ISO 与 IEC 保持密切合作关系。ISO 的工作语言是英文、法文和俄文,会址设在日内瓦。

ISO 的成员团体分正式成员和通讯成员。正式成员是指由各国最有代表性的标准化机构代表其国家或地区参加的成员,并且只允许每个国家有一个组织参加。通讯成员是尚未建立全国性标准化机构的国家,一般不参与 ISO 的技术工作,但可参会了解工作进展,当条件成熟时,可以通过一定程序成为正式成员。1947 年 ISO 成立时只有 25 个成员团体,但经过 50 年的发展,截至 1998 年有团体成员(国家标准化机构)122 个,其中正式成员 86 个,通讯成员 36 个。

成员全体大会是 ISO 的最高权力机构。理事会是 ISO 常务机构,由正、副主席、司库和 18 个理事国代表组成,每年召开一次会议,理事会成员任期 3 年,每年改选 1/3 的成员。理事会下设若干专门委员会,其中之一是技术委员会(TS),技术委员会完成 ISO 的技术工作。ISO 按专业性质设立技术委员会,各技术委员会根据工作需要可设立若干分委员会(SC),TC 和 SC 下面可设立若干工作组(WG)。TC 和 SC 的成员分为积极参加成员(P 成员)和观察员(O 成员)两种。P 成员可参与 TC 和 SC 的技术工作,而 O 成员则只能了解工作进度和得到技术组织的信息资料,不参加技术工作。每个 TC 或 SC 均从 P 成员中任命一个成员主持秘书处并领导该委员会或分委员会。ISO 现有技术组织 2871 个,其中技术委员会(TC)191 个,分技术委员会(SC)572 个,工作组 2063 个,临时专题小组 45 个。

2. 国际电工委员会 IEC(International Electrotechnical Commission, IEC)

国际电工委员会 IEC 成立于 1906 年,是世界上最早的非政府性国际电工标准化机

构,是联合国经社理事会(ECOSOC)的甲级咨询组织。自1947年ISO成立后,IEC曾作为一个电工部并入ISO,但在技术上和财务上仍保持独立。1976年双方又达成新协议,IEC从ISO中分离出来,两组织各自独立,自愿合作,互为补充,共同建立国际标准化体系,IEC负责有关电气工程及电子领域国际标准化工作,其他领域则由ISO负责。

IEC的工作领域包括电工领域各个方面,如电力、电子、电信和原子能方面的电工技术等。理事会是IEC的最高权力机构,会址设在日内瓦。IEC理事会下设执行委员会与合格评定局。执行委员会负责管理技术委员会(TC)和技术咨询委员会。合格评定局管理各认证委员会,在组织上自成体系。它是世界范围的自愿认证机构,其宗旨是促进国家或国际间的自由贸易。按照严格的认证程序,以国际标准为依据对电工产品生产厂的技术力量和管理水平实行全面的审核和评审;对要求认证的元器件按标准要求进行测试和检验。对符合质量要求的产品授予合格证书,以确保产品质量达到和保持标准要求的水平。

16.7.2 区域标准化组织

区域标准化组织是指同处一个地区的某些国家组成的标准化组织。区域是指世界上按地理、经济或民族利益划分的区域。参加组织的机构有的是政府性的,有的是非政府性的,为发展同一地区或毗邻国家间的经济及贸易,维护该地区国家的利益,协调本地区各国标准和技术规范而建立的标准化机构。其主要职能是制定、发布和协调该地区的标准。

(1) 欧洲标准化委员会(CEN)成立于1961年,由欧洲经济共同体(EEC)、欧洲自由联盟(EFTA)所属国家的标准化机构组成,主要任务是协调各成员国的标准,制定必要的欧洲标准(EN),实行区域认证制度。

(2) 欧洲电工标准化委员会(CEN EL EC)成立于1972年,由欧洲电工标准协调委员会(CEN EL)和欧洲电工协调委员会共同市场小组(CEN EL COM)合并组成,主要是协调各成员国电器和电子领域的标准,以及电子元器件质量认证,制定部分欧洲标准(EN)。

(3) 亚洲标准咨询委员会(ASAC)成立于1967年,由联合国亚洲与太平洋经社委员会协商建立,主要是在ISO和IEC标准的基础上,协调各成员国的标准化活动,制定区域性标准。

(4) 国际电信联盟ITU(International Telecommunication Union)于1865年5月在巴黎成立,1947年成为联合国的专门机构,是世界各国政府电信主管部门之间协调电信事务的一个国际组织。该组织研究制定有关电信业务的规章制度,通过决议提出推荐标准,收集有关情报。ITU的目的和任务是维持和发展国际合作,以改进和合理利用电信,促进技术设施的发展及有效应用,提高电信业务的效率。

16.7.3 行业标准化组织

行业标准化组织是指制定和公布适应某个业务领域标准的专业标准化团体,以及在其业务领域开展标准化工作的行业机构、学术团体或国防机构。

(1) 美国电气电子工程师学会 IEEE (Institute of Electrical and Electronics Engineers) 是由美国电气工程师学会(AIEE)和美国无线电工程师学会(IRE)于 1963 年合并而成的,是美国规模最大的专业学会。IEEE 主要制定电气与电子设备、试验方法、原器件、符号、定义以及测试方法等方面的标准。近年该学会专门成立了软件标准分技术委员会(CESS),积极开展软件标准化活动,取得了显著成果,受到了软件界的关注。IEEE 通过的标准常常要报请 ANSI 审批,使之具有国家标准的性质。因此,IEEE 公布的标准常冠有 ANSI 字头。例如,ANSI/IEEE Str 828—1983 软件配置管理计划标准。

(2) 美国国防部批准、颁布适用于美国军队内部使用的标准,代号为 DOD(采用公制计量单位的以 DOD 表示)和 MIL。

(3) 我国国防科学技术工业委员会批准、颁布适合于国防部门和军队使用的标准,代号为 GJB。例如,1988 年发布实施的 GJB473—88 军用软件开发规范。

16.7.4 国家标准化组织

国家标准化组织是指在国家范围内建立的标准化机构,以及政府确认(或承认)的标准化团体,或者接受政府标准化管理机构指导并具有权威性的民间标准化团体。

(1) 美国国家标准学会(American National Standards Institute, ANSI): ANSI 是非赢利性质的民间标准化团体,但它实际上已成为美国国家标准化中心,美国各界的标准化活动都围绕它开展。通过它使政府有关系统和民间系统相互配合,起到了政府和民间标准化系统之间的桥梁作用。ANSI 协调并指导美国全国的标准化活动,给标准制定、研究和使用单位以帮助,提供国内外标准化情报。ANSI 本身很少制定标准,主要是将其他专业标准化机构的标准经协商后冠以 ANSI 代号,成为美国国家标准。

(2) 英国标准学会(British Standards Institution, BSI): BSI 是世界上最早的全国性标准化机构,它是政府认可的、独立的、非赢利性民间标准化团体。主要任务是:为增产节约努力协调生产者和用户之间的关系,促进生产,达到标准化;制定和修订英国标准,并促进其贯彻执行;以学会名义,对各种标志进行登记,并颁发许可证;必要时采取各种行动,保护学会利益;对外代表英国参加国际或区域标准化活动。

(3) 德国标准化学会(Deutsches Institution fur Normung, DIN): DIN 始建于 1917 年,当时称为德国工业标准委员会(NADI),1926 年改为德国标准委员会(DNA),1975 年又改名为联邦德国标准化学会(DIN)。DIN 是一个注册的公益性民间标准化团体,前联

邦政府承认它为联邦德国和西柏林的标准化机构。德国统一后,其活动已遍及全德国。DIN 在国际上一直享有盛誉,其制定的标准在很多国家被广泛采用。

(4) 法国标准化协会(Association Francaise de Normalisation, AFNOR): AFNOR 成立于 1926 年,是一个公益性的民间团体,也是一个被政府承认,为国家服务的组织。1941 年 5 月 24 日颁布的一项法令确认 AFNOR 接受法国政府的标准化管理机构“标准化专署”指导,按政府指导开展工作,并定期向标准化专员汇报工作。AFNOR 负责标准的制定和修订工作,并宣传、出版和发行标准,实施产品质量认证。

16.8 ISO9000 标准简介

16.8.1 ISO9000 标准

ISO9000 标准是一系列标准的统称。ISO9000 系列标准由 ISO/TC176 制定。TC176 是 ISO 的第 176 个技术委员会(质量管理和质量保证技术委员会),专门负责制定质量管理和质量保证技术的标准。经过 TC176 多年的协调以及有关国家质量管理专家近 10 年的不懈努力,总结了美国、英国、加拿大等工业发达国家的质量保证技术实践经验,于 1986 年 6 月 15 日正式发布了 ISO8402《质量-术语》标准,又于 1987 年 3 月正式公布了 ISO9000~ISO9004 五项标准,这五项标准与 ISO8402: 1986 一起统称为 ISO9000: 1987 系列标准。随着工业、经济的不断发展,国际贸易交往日益频繁,产品质量成为各方面关注的焦点。因此,质量管理的对象也从硬件逐渐扩展到了硬件、软件、文档和服务。按照全面质量管理的 PDCA 工作模式,TC176 于 1990 年在第九届年会上提出了《90 年代国际质量标准的实施策略》,决定对 ISO9000 标准进行修改。通过对 ISO9001/2/3/4 标准技术内容的修订(局部修改),形成 1994 版。在 ISO9000-1: 1994 中增加了过程和过程网络等基本概念,为进一步修改提供了过渡性理论基础。之后,引入与 PDCA 循环模式统一的过程方法模式,从总体结构和原则到具体的技术内容对 ISO9000 系列标准进行了全面修改,形成 2000 年版。2000 年 12 月 15 日,ISO9000: 2000 系列标准正式发布实施。

ISO9000 系列标准的质量管理模式为企业注入新的活力和生机,为质量管理体系提供评价基础,为企业进行世界贸易带来质量可信度。从 ISO9000 系列标准的演变过程可见,ISO9001: 1987 系列标准从自我保证的角度出发,更多关注的是企业内部的质量管理和质量保证。ISO9001: 1994 系列标准则通过 20 个质量管理体系要素,把用户要求、法规要求及质量保证要求纳入标准的范围中。ISO9001: 2000 系列标准在标准构思和标准目的等方面体现了具有时代气息的变化,过程方法的概念,顾客需求的考虑,以及将持续改进的思想贯穿于整个标准,把组织的质量管理体系满足顾客要求的能力和程度

体现在标准的要求之中。

16.8.2 ISO9000:2000 系列标准文件结构

ISO9000:2000 系列标准现有 13 项标准,由 4 个核心标准,1 个支持标准,6 个技术报告,3 个小册子和 1 个技术规范构成,参见表 16-1。

表 16-1 ISO9000:2000 系列标准文件结构

核 心 标 准	ISO9000:2000《质量管理体系 基础和术语》
	ISO9001:2000《质量管理体系 要求》
	ISO9004:2000《质量管理体系 业绩改进指南》
	ISO19011:2000《质量管理体系和环境管理体系审核指南》
其 他 标 准	ISO10012《测量设备的质量保证要求》
技 术 报 告	ISO10006《项目管理指南》
	ISO10007《技术状态管理指南》
	ISO10013《质量管理体系文件指南》
	ISO10014《质量经济性指南》
	ISO10015《教育和培训指南》
	ISO10017《统计技术在 ISO9000 中的应用指南》
小 册 子	质量管理原理
	选择和使用指南
	小型企业的应用指南

16.8.3 ISO9000:2000 核心标准简介

1. ISO 9000:2000《质量管理体系 基础和术语》

该标准描述了质量管理体系的基础,并规定了质量管理体系的术语和基本原理。术语标准是讨论问题的前提,统一术语是为了明确概念,建立共同的语言。

该标准在总结了质量管理经验的基础上,明确了一个组织在实施质量管理中必须遵循的 8 项质量管理原则,也是 ISO9000:2000 系列标准制定的指导思想和理论基础。该标准提出的 10 个部分 87 个术语,在语言上强调采用非技术性语言,使所有潜在用户易于理解。为便于使用,在标准附录中,推荐了以“概念图”方式来描述相关术语的关系。

2. ISO 9001: 2000《质量管理体系 要求》

该标准提供了质量管理体系的要求,供组织证实其具有提供满足顾客要求和适用法规要求的产品的能力时使用。组织通过有效地实施质量管理体系,包括过程的持续改进和预防不合格的产品,使顾客满意。该标准是用于第三方认证的惟一质量管理体系要求标准,通常用于企业建立质量管理体系以及申请认证。它主要通过对申请认证组织的质量管理体系提出各项要求来规范组织的质量管理体系。主要分为质量管理体系、管理职责、资源管理、产品实现、测量分析和改进 5 大模块,构成一种过程方法模式的结构,符合 PDCA 循环规则,且通过持续改进的环节使质量管理体系的水平达到螺旋式上升的效果,其中每个模块中又有许多分条款。

3. ISO 9004: 2000《质量管理体系 业绩改进指南》

该标准给出了改进质量管理体系业绩的指南,描述了质量管理体系应包括持续改进的过程,强调通过改进过程,提高组织的业绩,使组织的顾客及其他相关方满意。

该标准是和 ISO9001: 2000 协调一致并可一起使用的质量管理体系标准。两个标准采用相同的原则,但应注意其适用范围不同,而且 ISO9004 标准不拟作为 ISO9001 标准的实施指南。通常情况下,当组织的管理者希望超越 ISO9001 标准的最低要求,追求增长的业绩改进时,一般以 ISO9004 标准为指南。

4. ISO 19011: 2001《质量管理体系和环境管理体系审核指南》

该标准提供了质量管理体系和环境管理体系审核的基本原则、审核方案的管理、环境质量管理体系的实施以及对环境和质量管理体系评审员资格的要求。

该标准是 ISO/TC176 与 ISO/TC207(环境管理技术委员会)联合制定的,按照“不同管理体系,可以共同管理和审核”的原则,在术语和内容方面兼容了质量管理体系和环境管理体系两方面的特点。

16.8.4 ISO9000: 2000 系列标准确认的 8 项原则

ISO9000 系列质量管理体系在 ISO9000: 2000 和 ISO9004: 2000 标准中提及的 8 项质量管理原则是该标准中一个非常重要的内容,是整个 ISO9000 系列质量管理体系标准的精髓和纲领。它是标准的理论基础,又是组织领导者进行质量管理的基本原则。增加了贯彻 8 项质量管理基本原则的要求,并且以此为主线,贯彻于各个过程并体现在标准的全部内容中。8 项质量管理原则如下所述。

1. 以顾客为中心

“组织依存于顾客。因此,组织应了解顾客当前和未来的需求,满足顾客要求并争取超越顾客期望。”顾客是每一个组织存在的基础,顾客的要求是第一位的,组织应全面地调查和研究顾客的需求和期望,并把它转化为质量要求,采取有效措施使其实现。例如,从

市场上获得顾客的质量评价,测量顾客的满意度,并根据结果来指导组织的运作;提高组织资源使用的有效性来增加顾客的满意度;确保顾客满意和其他相关方的利益(例如所有者、雇员、供方、金融机构、所在社区和整个社会)。

2. 领导作用

“领导者确立一致的组织宗旨和方向。他们应当创造并保持员工能够充分参与实现组织目标的内部环境。”领导者具有决策和领导一个组织的关键作用。为了营造一个良好的环境,领导者应关注顾客要求,清晰地规划组织的未来,建立质量方针和质量目标,确保建立和实施一个有效的质量管理体系,以统一的方式来评估、安排和实施活动,并随时将组织运行的结果与目标比较,根据情况决定实现质量方针和目标的措施,决定持续改进的措施。考虑所有利益相关方的需要,包括顾客、员工、供方、所在社区和整个社会。在组织的各个层次树立价值共享和道德伦理的观念,向员工提供所需的资源和培训,赋予他们履行职责和义务的自由度,鼓舞、奖励和承认员工的贡献,让员工了解组织目标并追求组织的成功。在领导作风上要做到透明、务实和以身作则。

3. 全员参与

“各级人员是组织之本,只有他们的充分参与,才能使他们的才干为组织带来最大的收益。”组织的质量管理不仅需要最高管理者的正确领导,还有赖于全员的参与。全体员工是每个组织的基础,让他们了解其贡献的重要性和在组织中的角色,承担起解决问题的责任,使得他们的才干为组织带来收益。所以要对职工进行质量意识、职业道德、以顾客为中心的意识和敬业精神的教育,激发他们的积极性和责任感,在团队(组织)中自由地分享知识和经验,公开地讨论问题和观点,主动地寻求机会来加强他们的技能、知识和经验,积极为组织的持续改进做出贡献,进一步促进实现组织目标的创新和创造力。

4. 过程方法

“将相关的资源和活动作为过程进行管理,可以更高效地得到期望的结果。”ISO 9000:2000 系列标准建立了一个过程模式,将“管理职责、资源管理、产品实现、测量及分析和改进”作为体系的4大主要过程,描述其相互关系、并以顾客要求为输入,提供给顾客的产品为输出,通过信息反馈来测定顾客满意度,评价质量管理体系的业绩。应用过程方法有助于不断改进,协调一致,并可获得预期的结果,以便提供重点及有优先次序的改进方案,达到有效地使用资源、降低成本、缩短周期和提高质量。过程方法的原则不仅适用于某些简单的过程,也适用于由许多过程构成的过程网络。

5. 管理的系统方法

“识别、理解和管理作为体系的相互关联的过程,有助于提高组织的有效性和效率。”针对设定的目标,识别、理解体系各个过程之间的内在关联性,采用科学的方法协调和整合过程。构造和实施质量管理体系的方法,既可用于新建体系,也可用于现有体系的改

进。这种方法的实施,能够深入理解为实现目标所必需的作用和责任;提供对过程能力及产品可靠性的信任;便于将注意力集中于重点过程;为持续改进打好基础;使顾客满意,使利益相关方对组织的协调性、有效性和效率建立信心,最终使组织获得成功。

6. 持续改进

“组织总体业绩的持续改进应是组织的一个永恒的目标。”在质量管理体系中,改进是指产品质量、过程及体系有效性和效率的提高。持续改进包括:了解现状;建立目标;寻找、评价和实施解决办法;测量、验证和分析结果,把更改纳入文件等活动。把产品、过程和体系的持续改进作为组织内每个成员的目标,根据组织的战略意图协调各层次上的改进活动,建立指导和测量跟踪持续改进的目标,提供持续改进的方法和工具,来持续改进组织的业绩。在竞争激烈的市场环境中,通过改进组织能力,可以对机遇快速灵活反应,增强竞争优势。

7. 基于事实的决策方法

有效的决策只能建立在对数据和信息进行分析的基础上。对数据和信息的逻辑分析或直觉判断是有效决策的基础。以事实为依据,参照实际记录来证明以往决策的有效性,对各种意见和决定加以评审、质疑和修改,可防止决策失误。以事实为依据,要求数据和信息具有足够的精确度、可靠性和可获取性,需要使用正确的方法分析数据和信息。统计技术是最重要的工具之一,可用来测量、分析和说明产品和过程的变异性,为持续改进的决策提供依据。

8. 互利的供求关系

供求双方是相互依存的,通过互利的关系可增强双方创造价值的能力。供方提供的产品将对组织向顾客提供满意的产品产生重要影响,因此处理好与供方的关系,影响到组织能否持续稳定地提供顾客满意的产品。对供方不能只讲控制(或提要求)不讲合作互利,需要在对短期收益和长期利益综合平衡的基础上建立相互关系,对市场或顾客需求和期望的变化,一起做出灵活快速的反应。特别是对关键的供方,更要注意建立合作关系,开展联合开发和改进活动,开放式地进行交流,激励和承认供方的改进和成就。建立互利关系,共享经验和资源,分享信息和对未来的规划,这对供求双方都有利。

16.9 能力成熟度模型 CMM 简介

软件质量是人们实践产物的属性和行为,是一个很复杂的事物性质和行为,可以通过一些方法和人们的活动,来改进质量。概括地说,通过控制软件生产过程、提高软件生产者的组织性和软件生产者的个人能力来改进软件质量。现有的软件质量改进方法有很多,软件能力成熟度模型(Capability Maturity Model, CMM)是一个目前国际上较流行、

ISO/IEC 15504 提供了一种有组织的、结构化的软件过程评估方法,以便实施软件过程的评估。在 ISO/IEC 15504 中定义的过程评估办法旨在为描述工程评估结果的通用方法提供一个基本原则,同时也对建立在不同但兼容的模型和方法上的评估进行比较。过程评估有两个主要的使用环境:软件过程改进或者软件过程能力评定。在软件过程改进环境中,过程评估提供了诸多方法,用于在企业内部根据所选择的过程的性能来描述当前实践的特性。根据企业的商业需要对结果进行分析以确定该过程内在的优点、不足和风险,从而可以判断过程是否有效地实现其目标;或者没有实现目标时质量低下、耗时过多、成本过高的主要原因。评估结果可以为软件过程改进的优先顺序提供相应的基础和依据。在确定软件过程能力的环境中,评估是通过与目标过程能力的对比分析,评估过程的能力,从而确定所评估的过程对实现有关项目的风险情况。在 ISO/IEC15504 文件中涉及了过程评估的各个方面,其文档主要包括以下几个部分。

1. 第 1 部分 概念和绪论指南

该部分给出了关于软件过程改进和过程评估概念以及在过程能力确定方面的总体信息。它描述了 ISO/IEC 15504 文档的各部分是如何组织在一起的,并为选择和使用各部分提供指南。此外,本部分还解释了 ISO/IEC 15504 中所包含的要求对执行评估的适用性、支持工具的建立与选择以及在附加过程的建立和发展方面所起的作用。

2. 第 2 部分 过程和过程能力参考模型

从内容上说,该部分是在比较高的层次上详细定义了一个用于过程评估的二维参考模型。此模型中描述了过程和过程能力。通过将过程中的特点与不同的能力等级进行比较,再利用此模型中定义的一系列过程和框架就可对过程能力加以评估。

3. 第 3 部分 实施评估

为了确保等级评定的一致性和可重复性(即标准化),ISO/IEC 15504 为软件过程评估提供了一个框架,并为进行评审提出了最低要求。这些要求有助于确保评估结果内在的一致性,为评级和验证与要求的一致性提供了依据。该部分以及与该部分有关的内容详细定义了实施评估时的要求,这样得到的评估结果才有可重复性、可信性以及可持续性。

4. 第 4 部分 评估实施指南

通过这部分内容,可以指导使用者如何进行软件过程评估。这个具有普遍意义的指导适用于所有企业,同时也适用于采用不同的方法、技术以及支持工具的过程评估。它包括如何选择并使用兼容的评估,如何选择用于支持评估的方法,如何选择适合于评估的工具与手段。该部分内容对过程评估做了概述,并且以指南的形式对用于评估的兼容模型、文件化的评估过程以及工具的使用和选择等方面的需求做了解释。

5. 第 5 部分 评估模型和标志指南

这部分内容为支持过程评估提出了一个评估模型的范例。此评估模型与第二部分所描述的参考模型相兼容,具体表述了任何兼容评估模型都期望具有的核心特征。该指南是以此评估模型中所包含的指示标志的形式给出的,这些指示标志不但可在过程改进程序中加以使用,还有助于评价和选择评估模型、方法或工具。采用这种方式并结合可靠的方法,有可能对过程能力做出一致的且可重复的评级

6. 第 6 部分 评估师能力指南

这部分提供了关于评估师进行软件过程评估的资格和准备的指南。它详细说明了一些可用于验证评估师胜任能力的方法,以及相应的教育、培训和经验,还包括可能用于验证胜任能力和证实受教育程度、培训情况和经验的一些机制。

7. 第 7 部分 过程改进应用指南

该部分提供了关于使用软件过程评估作为首要方法去理解一个企业软件过程的当前状态和使用评估结果去形成并优化改进方案方面的指南。该过程改进指南涉及过程改进综述、过程改进方法、文化问题、管理等专题,包括一个过程量度的总体框架。该指南用于指导在连续循环里进行软件过程改进时把软件过程评估当成改进框架和方法的一部分使用。一个企业可以根据其具体情况和需要从参考模型中选择所有的或部分软件过程用于评估或改进。

8. 第 8 部分 确定供方能力应用指南

这部分内容为以过程能力确定目的而进行的过程评审提供应用指南。它讲述了为对过程能力加以判断,应如何定义输入和如何运用评估结果。这不仅可直接用于对当前状况进行判断,而且也可以对复杂情况加以判断,例如对未来进行预测。该部分中关于过程能力的判断方法不仅适合于任何希望确定其自身软件过程的过程能力的企业,同样也适用于对供应商的能力进行判断。

9. 第 9 部分 词汇

本部分定义了 ISO/IEC TR 15504 整个技术报告中使用的术语。术语首先按字母顺序排列以便查阅;然后,再按逻辑类型进行分类以便于理解(将相关的术语安排在一类)。

全国计算机技术与软件专业技术资格（水平）考试指定用书

根据人事部、信息产业部文件，计算机技术与软件专业技术资格（水平）考试纳入全国专业技术人员职业资格证书制度的统一规划。通过考试获得证书的人员，表明其已具备从事相应专业岗位工作的水平和能力，用人单位可根据工作需要从获得证书的人员中择优聘任相应专业技术职务（技术员、助理工程师、工程师、高级工程师）。计算机技术与软件专业实施全国统一考试后，不再进行相应专业技术职务任职资格的评审工作。

ISBN 7-302-09096-3



9 787302 090960 >

定价：66.00元

www.TopSage.com